Opinnäytetyö AMK

Tieto- ja viestintätekniikka

2018

Joni Markkula

LIIKEANTURIN HYÖDYNTÄMINEN OPTISESSA SOVELLUKSESSA

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ AMK | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2018 | 52 sivua, 10 liitesivua

Joni Markkula

LIIKEANTURIN HYÖDYNTÄMINEN OPTISESSA SOVELLUKSESSA

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa laitteisto, joka indikoi käyttäjälle LSM6DSM-liikeanturilta luettavia kiihtyvyys- ja kulmanopeusarvoja sekä näistä johdettuja tuloksia. Laitteiston vaatimusmäärittelyksi asetettiin indikoida käyttäjälle laitteeseen kohdistuvaa negatiivista kiihtyvyyttä eli liikkeen hidastumista. Työn vaatimusmäärittelyn mukainen laitteisto saavutettiin toteuttamalla sulautettu ohjelmisto laitteen LPC11U14FBD48-mikrokontrollerille.

LSM6DSM on 6-akselinen inertiamoduuli, jossa yhdistyvät 3-akselinen MEMS-kiihtyvyysanturi ja 3-akselinen MEMS-gyroskooppi eli kulmanopeusanturi. Mikrokontrollerin ja liikeanturin tiedonsiirtoon käytettiin I²C-väylä topologiaa, jota hyödyntäen liikeanturilta saatiin luettua kiihtyvyys- ja kulmanopeusarvoja. Liikeanturilta luettavien arvojen ja näistä johdettujen tuloksien indikoimiseen käytettiin laitteistoon integroituja ledejä. Laitteiston sulautettu ohjelmisto kirjoitettiin käyttäen laiteläheistä C-kieltä. Työssä syvennyttiin liikeanturin ja mikrokontrollerin sisältämään rajapintaan ja toimintaperiaatteeseen sekä kokonaisuudessaan laitteiston sisältämään elektroniikkaan ja sulautetun ohjelmiston toteuttamiseen niin teorian kuin käytännön osalta.

Opinnäytetyön tuloksena syntyi laitteiston vaatimusmäärittelyn mukainen prototyyppi käytettäessä laitetta stabiilissa kohdeympäristössä. Työn tuloksena syntyneen prototyypin pohjalta voidaan päätellä, että MEMS-teknologiaan perustuvalta LSM6DSM-liikeanturilta voidaan lukea kiihtyvyys- ja kulmanopeusarvoja kohtalaisen tarkasti.

ASIASANAT:

Mikrokontrolleri, kiihtyvyysanturi, kulmanopeusanturi, ohjelmointi

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2018 | 52 pages, 10 pages in appendices

Joni Markkula

UTILIZATION OF ACCELEROMETER AND GYROSCOPE IN OPTICAL APPLICATION

The aim of this thesis was to design and implement a piece of hardware that indicates acceleration and angular velocity values read from the LSM6DSM motion sensor and the results derived from them. The hardware specification was set to indicate to a user the negative acceleration of the device, i.e., deceleration of the device. The hardware specification was achieved by implementing a program of embedded software to LPC11U14FBD48 microcontroller.

LSM6DSM is 6-axis inertial module combining a 3-axis MEMS accelerometer and a 3-axis MEMS gyroscope – an angular rate sensor. The I²C bus topology was used for the data transfer between the microcontroller and the motion sensor. By utilizing the I²C bus, the motion sensor's acceleration and angular velocity values were transferred to the microcontroller. The hardware integrated LEDs were used to indicate the motion sensor read values and their derived results. The embedded software of the hardware was written using embedded C. The hardware's electronics and embedded software are explained in depth in this thesis both in theory and in practice.

As a result of this thesis project, a prototype according to the hardware specification was created when the device was used in a stable target environment. Based on the resulting prototype, it can be concluded that the LSM6DSM motion sensor based on the MEMS technology can read the acceleration and angular velocity values fairly accurately.

KEYWORDS:

Microcontroller, accelerometer, angular rate sensor, programming

SISÄLTÖ

SANASTO **6**

1 JOHDANTO **7**

2 KOHDEYMPÄRISTÖ **8**

2.1 Mikrokontrolleri 8

2.1.1 LPC11U14FBD48-mikrokontrolleri 8

2.2 Kiihtyvyysanturi 10

2.2.1 Kiihtyvyysanturin toimintaperiaate 10

2.3 Gyroskooppi 13

2.3.1 Gyroskoopin toimintaperiaate 13

2.4 LSM6DSM-liikeanturi 15

2.5 Output-datarekisteri 17

3 I²C-VÄYLÄ **21**

3.1 LSM6DSM-liikeanturin ja LPC11U14FBD48-mikrokontrollerin välinen kommunikointi 23

3.1.1 Liikeanturin rekisterin lukeminen 24

3.1.2 Liikeanturin rekisteriin kirjoittaminen 25

4 KEHITYSYMPÄRISTÖ **26**

4.1 LPCXpresso 26

4.2 LPCXpresso-alusta 26

4.3 C-kieli 28

5 PROTOTYYPIN SUUNNITTELU JA TOTEUTUS **29**

5.1 Vaatimusmäärittely 29

5.2 Projektin luominen 29

5.3 Kytkentä 31

5.3.1 Mikrokontrolleri 32

5.3.2 Liikeanturi 35

5.3.3 LED 36

5.4 Sulautettu ohjelmisto 37

5.4.1 Arkkitehtuuri 37

5.4.2 I²C-väylä 38

5.4.3 Kiihtyvyysarvojen lukeminen ja prosessointi 40
5.4.4 Kulmanopeusarvojen lukeminen ja prosessointi 43
5.4.5 Liikeanturin kalibrointi 44
5.4.6 I/O-porttien initialisointi 46
5.5 Testaus 48

6 YHTEENVETO **51**

LÄHTEET **52**

LIITTEET

Liite 1. Sulautetun ohjelmiston lähdekoodi.

SANASTO

ω	kulmanopeus
a	kiihtyvyys
F	voima
g	g-voima
m	massa
debuggeri	Ohjelma, jota käytetään ohjelmiston tai laitteen testaamiseen ja virheiden etsimiseen.
dps	degrees per second, kulmanopeuden mittayksikkö (astetta/sekunti).
FLASH	Puolijohdemuisti, joka voidaan sähköisesti tyhjentää ja uudelleen ohjelmoida.
JTAG	Ohjelmiston ja laitteiston testaukseen käytettävä väylä.
I²C	Kaksisuuntainen ohjaus- ja tiedonsiirtoväylä.
I/O	Input/Output, siirräntä, tiedon siirtämistä tietokoneen komponenttien välillä.
MEMS	Micro Electro Mechanical Systems, mikrosysteemi.
ROM	Read Only Memory, lukumuisti.
SRAM	Static Random Access Memory, staattinen hajasaantimuisti, jota voidaan kirjoittaa ja lukea.

1 JOHDANTO

Puolijohdeteollisuuden kehitys viime vuosikymmenien aikana on mahdollistanut kustannustehokkaiden MEMS-anturien valmistamisen niin kiihtyvyys- kuin kulmanopeusanturien osalta. MEMS-anturien pieni koko mahdollistaa anturitiedon yhdistämisen sulautetun ohjelmiston kanssa mihin tahansa järjestelmään, näin luoden yhä innovatiivisimpia laitteita. Pienen vasteajan mahdollistavien, edullisen hinnan ja luotettavien mikroanturien johdosta myös autoteollisuus on suosinut kiihtyvyyttä mittaavia mikroantureita autojen turvatyynyjen laukaisemiseen törmäyksen sattuessa. Muita nykypäivän arkisia sovelluskohteita ovat esimerkiksi älypuhelimet, askelmittarit, kannettavat tietokoneet ja kamerat.

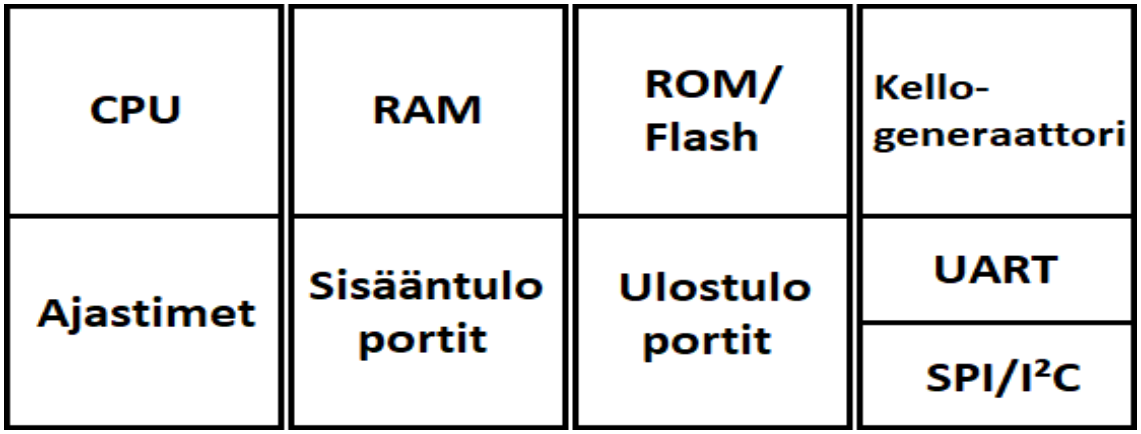
Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa laitteisto, joka indikoi käyttäjälle liikeanturilta luettavia kolmiakselisia kiihtyvyys- ja kulmanopeusarvoja sekä näiden tuloksia. Tarkemmin kuvailtuna laitteen tulee indikoida käyttäjälle ainoastaan laitteeseen kohdistuvaa negatiivista kiihtyvyyttä eli liikkeen hidastumista. Työn tavoitteeksi asetettiin edellä mainitun kuvauksen mukaisen laitteiston luominen prototyypiksi.

Opinnäytetyössä käsitellään prototyypin kehittämistä ja testaamista hyödyntäen MEMS-liikeanturia ja mikrokontrolleria, jotka mahdollistavat laitteiston kuvauksen mukaisten reunaehtojen täyttymisen. Työssä syvennytään lisäksi laitteen teknisiin ratkaisuihin niin sulautetun ohjelmiston kuin elektroniikan osalta.

2 KOHDEYMPÄRISTÖ

2.1 Mikrokontrolleri

Mikrokontrolleri on pieni tietokone, joka on yhtenäistetty ainoastaan yhden mikropiirin kanssa. Mikrokontrolleri pitää sisällään ainakin yhden tai useamman suorittimen, erityyppisiä muisteja ja ohjelmoitavia tulo- ja lähtöportteja. Mikrokontrollerit ovatkin nykypäivän sulautettujen laitteiden äly ja niitä löytyy esimerkiksi puhelimista, pyykinpesukoneista, henkilöautoista, televisioista yms. Kuvassa 1 on esitetty tyypillisen mikrokontrollerin rakenne.



Kuva 1. Tyypillinen mikrokontrollerin rakenne.

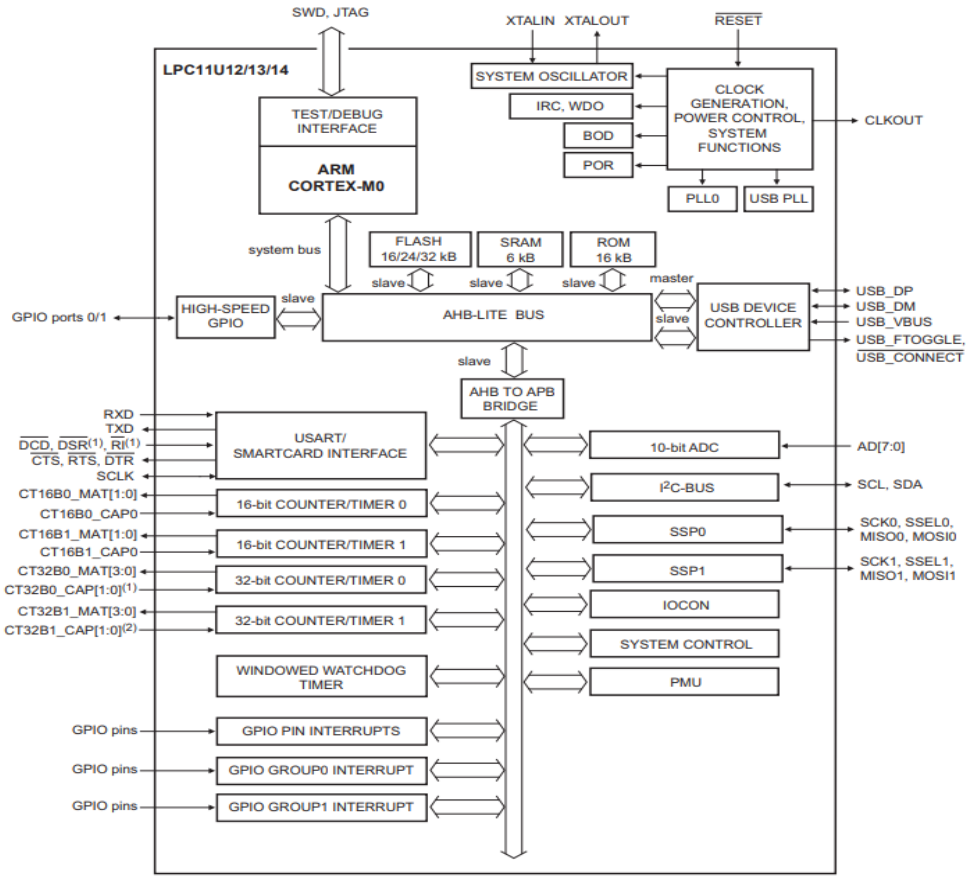
2.1.1 LPC11U14FBD48-mikrokontrolleri

Insinöörityön mikrokontrolleriksi valittiin LPC11U14FBD48, joka on ARM Cortex-M0 suorittimeen perustuva 32-bittinen mikrokontrolleri. Mikrokontrollerin valmistajana toimii puolijohteita valmistava suuryritys NXP Semiconductors. LPC11U14FBD48 sisältää JTAG ohjelmointi- ja debug-rajapinnan, jonka avulla mikrokontrolleria päästiin ohjelmoimaan ja testaamaan.

Työn kannalta LPC11U14FBD48-mikrokontrollerin tärkeisiin ominaisuuksiin kuuluu 50 MHz:n ARM Cortex-M0 suorittimen lisäksi:

- 32 kB Flash-muisti, johon tallennetaan laitteen suoritettava ohjelma.
- 6 kB SRAM-muisti, johon tallennetaan suoritettavan ohjelman paikalliset ja globaalit muuttujat, joiden arvo muuttuu suoritettavan ohjelman edetessä.
- 16 kB boot ROM, joka sisältää alkulatausohjelman mikä ajetaan prosessorin toimesta mikrokontrollerin käynnistyessä. Alkulatausohjelma sisältää laiteohjelmiston, joka mahdollistaa suoritettavan ohjelman suorittamisen Flash-muistista.
- 40 yleiskäyttöistä sisään- tai ulostuloporttia. Portteja käytettiin työssä ledien päälle tai pois kytkemiseen.
- I²C-väylä, jota hyödyntäen saadaan luettua kiihtyvyys- ja kulmanopeusarvoja LSM6DSM-liikeanturilta.

Kuvassa 2 on esitettynä LPC11U14FBD48-mikrokontrollerin lohkokaavio ja liitäntänastat. Tässä työssä ei käsitellä erikseen kaikkia mikrokontrollerin yksittäisiä lohkoja.



Kuva 2. LPC11U14FBD48-mikrokontrollerin lohkokaavio (NXP Semiconductors 2014).

2.2 Kiihtyvyysanturi

Kiihtyvyysanturi on sähkömekaaninen laite, joka mittaa kappaleeseen kohdistuvia kiihtyvyysvoimia. Kiihtyvyysvoimat voivat olla staattisia, kuten Maan vetovoima tai dynaamisia kuten kappaleeseen kohdistuva tärinä tai liike. Kiihtyvyysanturi mittaa kiihtyvyyttä metriä neliösekunnissa $(\mathrm{m/s^2})$ tai G-voimissa (g), missä $g = 9.8\ \mathrm{m/s^2}$. Kiihtyvyys tarkoittaa nopeuden muuttumista tietyssä ajassa. Kiihtyvyys voi olla positiivista tai negatiivista. Positiivinen kiihtyvyys tarkoittaa kappaleen nopeuden kasvua. Negatiivinen kiihtyvyys esiintyy, kun kappaleen negatiivinen nopeus jatkaa kasvamistaan tai kappaleen alun perin positiivinen nopeus hidastuu, kunnes kappaleen liikesuunta kääntyy ja negatiivinen nopeus alkaa kasvamaan. Kiihtyvyysanturit voivat mitata kiihtyvyyttä yhden, kahden tai kolmen akselin osalta X, Y ja Z. Kiihtyvyysanturien yleisiä käyttösovelluksia ovat esimerkiksi laitteen asennon selvittäminen Maahan nähden tai laitteen liikkumisen havainnollistaminen.

2.2.1 Kiihtyvyysanturin toimintaperiaate

Kiihtyvyysanturin toiminta perustuu yleisesti pietsoresistiiviseen-, pietsosähköiseen- tai kapasitiiviseen mittaukseen (Dimension Engineering LLC 2018). Tässä työssä käydään läpi kapasitiivinen toimintatapa, koska työssä käytetyn kiihtyvyysanturin toiminta perustuu kyseiseen mittaustapaan.

Newtonin toisen lain mukaan kappaleeseen vaikuttava kokonaisvoima F antaa m-massaiselle kappaleelle kiihtyvyyden a:

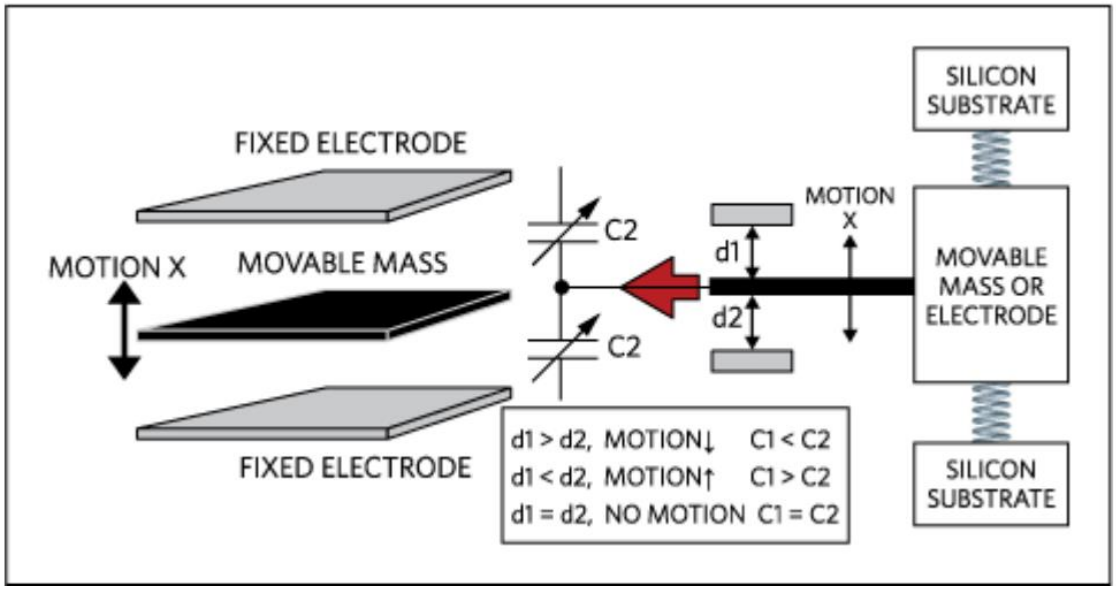
$$F = ma \tag{1}$$

Kiihtyvyysanturin toiminta perustuu kiihtyvyyden luomaan voimaan, jonka kiihtyvyysanturi mittaa. Varsinaisesti kiihtyvyysanturi ei mittaa siis kiihtyvyyttä, vaan kappaleeseen kohdistuvaa voimaa, jonka kiihtyvyys aiheuttaa kappaleeseen.

Kapasitiivinen mittaustapa perustuu kapasitanssin muutoksiin tunnetun massan liikkuessa kahden elektrodilevyn välissä. Kapasitiiviseen mittaustapaan perustuvat kiihtyvyysanturit ovat yleisimmin käytetty anturityyppi, koska ne ovat luotettavia, stabiileita,

toimivat alhaisella virrankulutuksella sekä niissä on hyvä kohinasuhde. (Dadafshar 2014.)

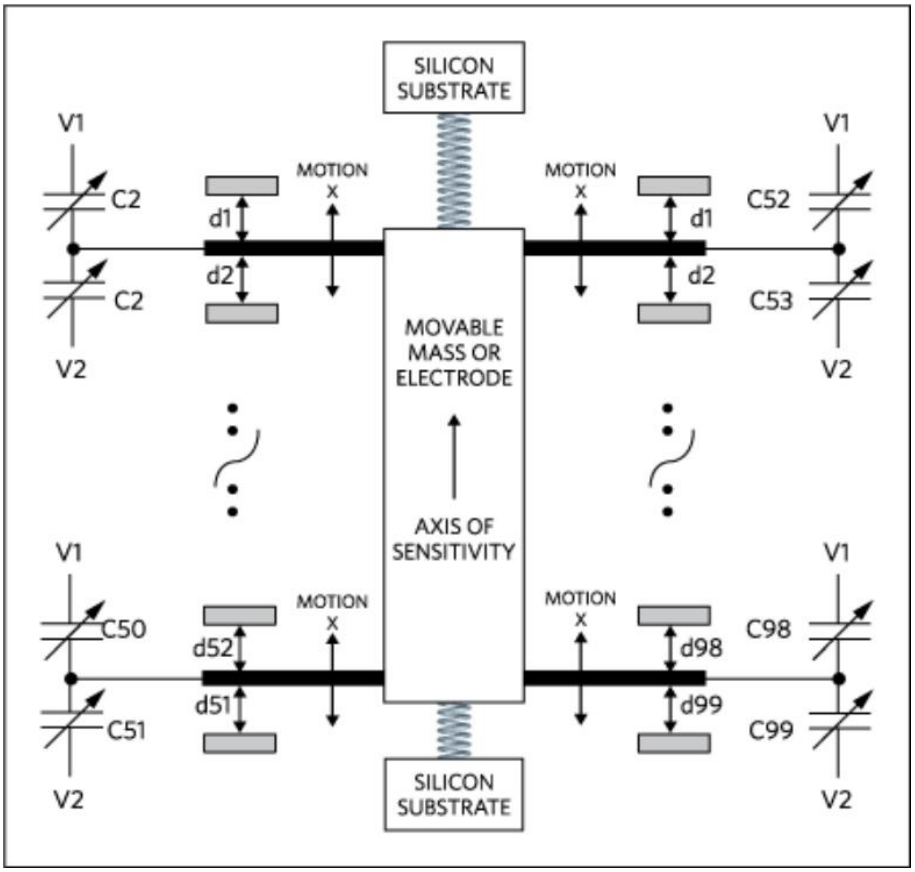
Kuvassa 3 on esitetty kapasitiiviseen mittaustapaan perustuvan kiihtyvyysanturin toimintaperiaate. Se koostuu yhdestä liikkuvasta massasta, joka mekaanisesti kiinnitetään jousen päähän ja sijoitetaan kahden elektrodilevyn väliin. Kiihtyvyyden muutos saa jousen päässä olevan massan liikkumaan (Motion X) lähemmäksi toista elektrodilevyä (d1 ja d2), näin muuttaen kondensaattoreiden kapasitanssia (C1 ja C2). Laskemalla kapasitanssien eron kondensaattoreiden välillä voidaan määritellä massan liikkumisen suunta.



Kuva 3. Kapasitiivisen kiihtyvyysanturin toimintaperiaate (Dadafshar 2014).

Yhden liikkuvan massan aiheuttama kapasitanssin muutos kondensaattoreiden välillä on äärimmäisen pieni, mistä on vaikea havaita todellista kiihtyvyyden muutosta. Tästä johtuen kapasitiiviseen toimintatapaan perustuvissa kiihtyvyysantureissa käytetään useita liikkuvia massoja elektrodilevyjen välillä, jotka ovat kytketty rinnakkain anturin rajapintaan. Kyseinen menetelmä aiheuttaa suuremman ja tarkemman kapasitanssin muutoksen havaitsemisen, joka tekee kyseisestä tekniikasta toteuttamiskelpoisen. (Dadafshar 2014.) Kuvassa 4 on esitetty useaan liikkuvaan massaan perustuva kapasitiivinen kiihtyvyysanturi. Toimintaperiaate on sama kuin yksittäisen massan liikkuessa, mutta sijoittamalla usea elektrodilevy rinnakkain saadaan paljon suurempi kapasitanssin muutos

aikaiseksi kiihtyvyyden muutoksen aikana. Kuvassa 4 V1 ja V2 ovat sähköisiä liitäntäpisteitä kummallakin puolella kondensaattoreita, jotka muodostavat jännitteenjakajan, jossa keskipisteenä on liikkuva massa.



Kuva 4. Kapasitiivinen kiihtyvyysanturi usealla liikkuvalla massalla (Dadafshar 2014).

Massan sijaintia vastaava analoginen jännite muutetaan digitaaliseksi sähkösignaaliksi AD-muuntimen avulla. AD-muuntimen kääntämä digitaalinen data voidaan näin siirtää I²C-väylää pitkin liikeanturin ja mikrokontrollerin välillä.

2.3 Gyroskooppi

Gyroskooppi eli kulmanopeusanturi on laite, joka mittaa kappaleen kulmanopeutta ω. Gyroskoopit voivat mitata kulmanopeutta yhden, kahden tai kolmen akselin osalta. Gyroskooppi mittaa kulmanopeutta yksikössä astetta/sekunti $(°/s)$ eli dps. Gyroskooppia käytetään kappaleen asennon mittaamiseen ja vakauttamiseen. Gyroskoopit ovat kehittyneet mekaanisista malleista, jotka muodostuivat pyörivästä pyörästä sekä kardaanisesti kiinnitetystä rengassysteemistä. Nykyisin gyroskoopit ovat erilaisia elektronisia ja optisia laitteita. (Sensorwiki 2016.)

Gyroskoopit voidaan jakaa kolmeen eri kategoriaan:

- Mekaaniset gyroskoopit
- Värähtelevään massaan perustuvat gyroskoopit
- Optiset gyroskoopit

Tässä työssä perehdytään värähtelevään massaan perustuvan gyroskoopin toimintaan, koska työn toteutukseen käytetty MEMS-liikeanturin gyroskooppi toimii kyseisellä menetelmällä.

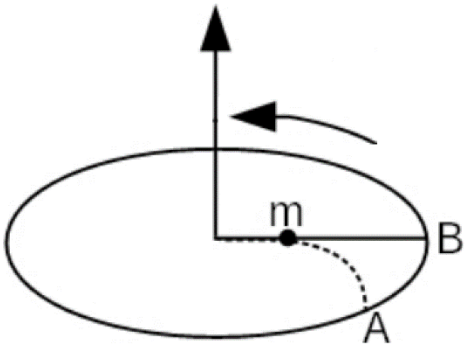
2.3.1 Gyroskoopin toimintaperiaate

Värähtelevään massaan perustuvan gyroskoopin toiminta perustuu Newtonin II lakiin niin kuin kiihtyvyysanturinkin. Kun pyörimisliike lisätään Newtonin yhtälöön, syntyy matemaattinen termi ns. virtuaalinen voima jota kutsutaan coriolisvoimaksi. Gyroskoopin toiminta perustuukin siihen, että pyörimisliike muutetaan coriolisvoimaksi. (Sensorwiki 2016.) Coriolisvoima voidaan esittää yhtälönä:

$$\overline{F_c} = 2m(\bar{v} \times \bar{\omega}), \quad (2)$$

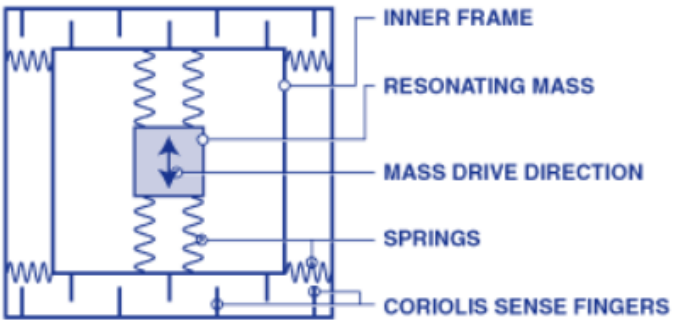
missä $\overline{F_c}$ on coriolisvoima ja $\bar{v}$ nopeus (Trusov 2011).

Coriolis-ilmiö syntyy silloin, kun jokin massa liikkuu pyörivässä ympäristössä. Esimerkkinä kuviossa 1 m-massa liikkuu keskeltä kohti pistettä B. Saapuessaan reunalle se saavuttaa pisteen A, eikä pistettä B johtuen coriolisvoimasta.



Kuvio 1. Coriolis-ilmiön periaate.

Värähtelevään massaan perustuva gyroskooppi sisältää massan, joka on yhdistetty sisäkehykseen jousilla. Sisäkehys on yhdistetty ulkokehykseen jousilla, jotka ovat 90° asteen kulmassa sisäkehyksen massan liikkumissuuntaan nähden. (Sensorwiki 2016.) Sisäkehyksen sisällä oleva massa on koko ajan kuvan 5 esittämän suunnan mukaisessa liikkeessä. Kappaletta pyöräyttäessä altistuu sisäkehys coriolis-ilmiölle, pakottaen sen sivuttaisliikkeeseen eli ulkokehyksen jousien suuntaan. Kyseisellä menetelmällä voidaan mitata coriolisvoiman aiheuttama kiihtyvyys kapasitanssin muutoksena elektrodilevyjen välissä, jotka ovat liitetty sisä- ja ulkokehyksiin. Kiihtyvyyden tuottama analoginen jännite voidaan muuttaa digitaaliseksi sähkösignaaliksi ja siirtää I²C-väylää pitkin liikeanturin ja mikrokontrollerin välillä. Kyseisellä menetelmällä saadaan mitattua tietyn akselin kulmanopeus ja sen suunta. Kuvassa 5 on esitetty värähtelevään massaan perustuvan gyroskoopin rakenne.



Kuva 5. Värähtelevään massaan perustuvan gyroskoopin rakenne (Sensorwiki 2016).

2.4 LSM6DSM-liikeanturi

LSM6DSM-liikeanturi on 6-akselinen inertiamoduuli, jossa yhdistyvät 3-akselinen MEMS-kiihtyvyysanturi ja 3-akselinen MEMS-gyroskooppi eli kulmanopeusanturi. Liikeanturi sisältää myös lämpötila-anturin. Liikeanturin valmistajana toimii puolijohteitavalmistava STMicroelectronics. (STMicroelectronics 2017.)

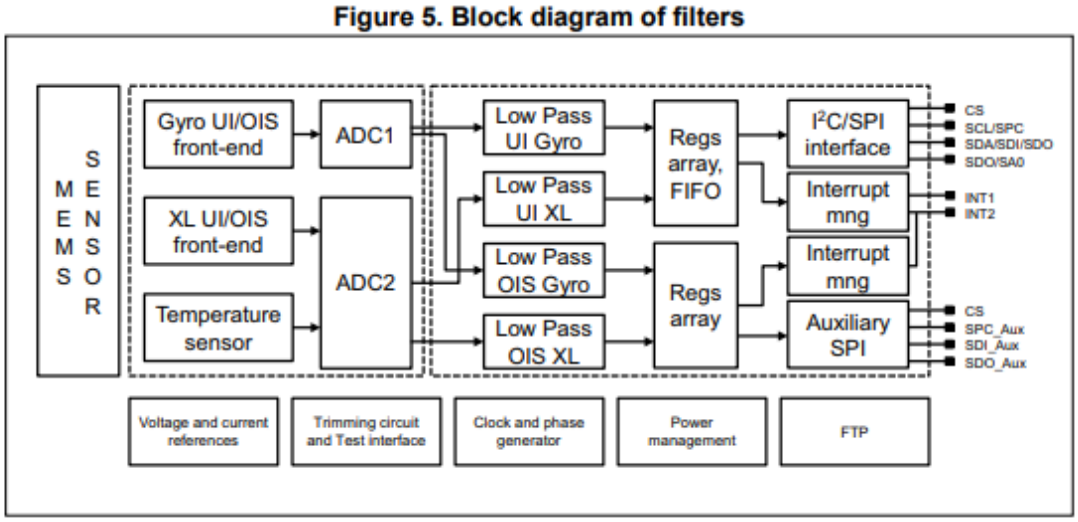
Liikeanturin 3-akselinen kiihtyvyysanturi mittaa kappaleen kiihtyvyyttä G-voimissa (g), missä $g = 9.8\ \mathrm{m/s^2}$. Liikeanturi on liitetty prototyyppiin siten, että Maan vetovoiman aiheuttama staattinen voima havaitaan kiihtyvyysanturin X-akselilla $1\ g$ voimana, prototyypin ollessa tasaisesti paikallaan. Kiihtyvyysanturin mittaaman yhden akselin kiihtyvyys voidaan ohjelmoida seuraaviin raja-arvoihin $\pm 2/\pm 4/\pm 8/\pm 16\ g$ (STMicroelectronics 2017). Tässä työssä kiihtyvyysanturin mittaaman yhden akselin kiihtyvyyden alueeksi ohjelmoitiin $\pm 2\ g$.

Liikeanturin 3-akselinen kulmanopeusanturi mittaa kappaleeseen kohdistuvaa kulmanopeutta ω, dps-yksiköinä. Kulmanopeusanturin mittaaman yhden akselin kulmanopeus voidaan ohjelmoida viiteen eri alueeseen $\pm 125/\pm 250/\pm 500/\pm 1000/\pm 2000\ \mathrm{dps}$ (STMicroelectronics 2017). Tässä työssä alueeksi ohjelmoitiin $\pm 2000\ \mathrm{dps}$.

Kiihtyvyys-, kulmanopeus- ja lämpötila-anturin mittaamat tulokset ajetaan ohjelmoitavien AD-muuntimien läpi. Kuvasta 6 nähdään, että kiihtyvyys- ja lämpötila-anturilla on yhteinen AD-muunnin ja kulmanopeusanturilla omansa. AD-muunnin muuttaa antureiden mittaaman analogisen jännitteen digitaaliseksi sähkösignaaliksi eli lukuarvoksi. AD-muuntimen muuttama digitaalinen sähkösignaali ajetaan alipäästösuodattimen läpi.

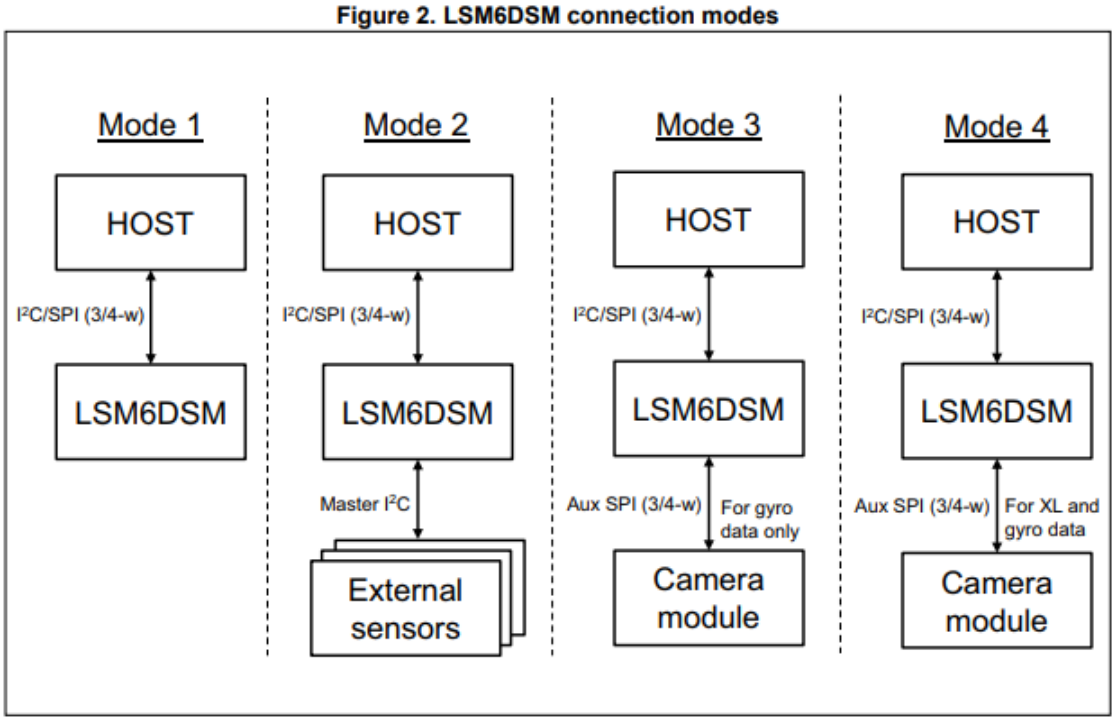
Kulmanopeus- ja kiihtyvyysanturin digitaalisen sähkösignaalin alipäästösuodatus riippuu liikeanturin elektronisesta kytkennästä (STMicroelectronics 2017). Liikeanturi voidaan kytkeä neljällä eri tavalla. Liikeanturin toiminallisuus riippuu sen kytkentätavasta, tähän asiaan perehdytään myöhemmin tässä luvussa.

Alipäästösuodattamisen jälkeen mittaustulokset puskuroidaan FIFO-puskureihin ja siirretään datarekistereihin. Mittaustulokset voidaan lukea suoraan liikeanturin rekisteristä tai FIFO-puskurista käyttämällä SPI- tai I²C-tiedonsiirtotekniikoita. Tässä työssä käytettiin I²C-väylää mikrokontrollerin ja liikeanturin väliseen kommunikointiin. I²C-väylän avulla voitiin kiihtyvyys- ja kulmanopeusarvot lukea suoraan liikeanturin rekistereistä.



Kuva 6. LSM6DSM-liikeanturin lohkokaavio (STMicroelectronics).

Liikeanturi voidaan kytkeä neljällä eri tavalla elektronisesti, mahdollistaen liikeanturille erilaisia toiminnallisuuksia. Kuvassa 7 on esitetty neljän eri tilan eroavaisuudet liikeanturin toiminnallisuuksissa. Tilassa 1 LSM6DSM-liikeanturia voidaan käyttää orjana I²C-väylässä tai SPI-väylässä. Tilassa 2 liikeanturia voidaan käyttää samoilla toiminnallisuuksilla kuin tilassa 1, mutta liikeanturiin voidaan liittää ulkoinen sensori I²C-väylää käyttäen, jossa liikeanturi toimii isäntälaitteena. Tilassa 3 liikeanturi toimii samoilla toiminnallisuuksilla kuin tilassa 1, mutta liikeanturiin voidaan liittää ulkoinen sensori (kuvassa 7 esimerkkinä on käytetty kameramoduulia) SPI-väylää käyttäen. Ulkoiselle sensorille voidaan välittää kulmanopeusanturin mittaamia kulmanopeusarvoja SPI-väylää hyödyntäen. Tilassa 4 liikeanturi toimii samalla periaatteella kuin tilassa 3, mutta ulkoiselle sensorille voidaan SPI-väylää hyödyntäen välittää kulmanopeusarvojen lisäksi kiihtyvyysanturin mittaamia kiihtyvyysarvoja. (STMicroelectronics 2017.) Tässä työssä LSM6DSM-liikeanturi kytkettiin elektronisesti tilaan 1, koska ulkoiselle sensorille ei ollut työssä tarvetta. Työn elektroniseen kytkentään perehdytään paremmin luvussa 5.3 Kytkentä.



Kuva 7. Liikeanturin neljä eri tilaa (STMicroelectronics 2017).

2.5 Output-datarekisteri

LSM6DSM-liikeanturin rekisterit ovat pituudeltaan 8-bittisiä. Rekisterit ovat joko luettavia (r) tai kirjoitettavia (w) tai molempia (r/w). Jokaisella rekisterillä on lisäksi oma 7-bittinen osoite, millä liikeanturin rekisterit erotellaan toisistaan. (STMicroelectronics 2017.) Osa rekistereistä kuitenkin mielletään pituudeltaan 16-bittisiksi. 16-bittinen rekisteri kuitenkin todellisuudessa muodostuu kahdesta 8 bitin rekisteristä. Rekisterin ollessa pituudeltaan 16-bittinen on rekisterin sisältämä 16-bittinen sana jaettu kahteen eri rekisteriin, näin muodostuen kahden komplementti luvusta. Hyvänä esimerkkinä toimivat kiihtyvyys- ja kulmanopeusanturin eri akselien rekisterit. Kuvassa 8 on esitetty liikeanturin X-akselin alempi rekisteri ja kuvassa 9 X-akselin ylempi rekisteri. Liikeanturin mitatessa X-akselin kiihtyvyyttä tallennetaan mitattu tulos 16-bittisenä sanana kahteen eri rekisteriin kahden komplementti lukuna. Alempi rekisteri sisältää 16-bittisen sanan 8 vähiten merkitsevää bittiä D0-D7 (LSB) ja ylempi rekisteri 8 eniten merkitsevää bittiä D7-D15.

Table 106. OUTX_L_XL register

D7	D6	D5	D4	D3	D2	D1	D0

Table 107. OUTX_L_XL register description

D[7:0]	X-axis linear acceleration value (LSbyte)

Kuva 8. Liikeanturin X-akselin alempi rekisteri (STMicroelectronics 2017).

Table 108. OUTX_H_XL register

D15	D14	D13	D12	D11	D10	D9	D8

Table 109. OUTX_H_XL register description

D[15:8]	X-axis linear acceleration value (MSbyte)

Kuva 9. Liikeanturin X-akselin ylempi rekisteri (STMicroelectronics 2017).

16-bittinen sana saadaan lukemalla molemmat rekisterit ja yhdistämällä rekisterien sisältämät kaksi tavua yhdeksi kokonaiseksi 16-bittiseksi sanaksi. Eniten merkitsevät bitit D15-D8 ovat sanassa ensimmäisenä ja vähiten merkitsevät bitit D7-D0 eniten merkitsevien bittien jälkeen. Kuvassa 10 on esitetty 16-bittinen sana, joka on muodostettu alemmasta ja ylemmästä rekisteristä.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Kuva 10. Yhtenäistetty 16-bittinen sana.

Kiihtyvyys- ja kulmanopeusanturin akselien X, Y ja Z mitatut tulokset tallennetaan kahteen eri rekisteriin kahden komplementtilukuna. Kiihtyvyys- ja kulmanopeusanturin mittaama 16-bittinen sana on raakadataa (raw data), mikä muodostuu mitatun akselin kiihtyvyydestä tai kulmanopeudesta. Jotta raakadata saataisiin oikeaan muotoon eli G-voimiksi g tai kulmanopeudeksi dps, tulee sille suorittaa matemaattisia operaatioita.

Kuvasta 11 nähdään LSM6DSM-liikeanturin kiihtyvyys- ja kulmanopeusanturin herkkyysasteet eri g ja dps tasoissa. Työssä kiihtyvyysanturi ohjelmoitiin tasoon $\pm 2\ g$ ja kulmanopeusanturi tasoon $\pm 2000\ \mathrm{dps}$. Työn kiihtyvyysanturin herkkyysaste on merkitty sinisellä ja kulmanopeusanturin punaisella korostuksella kuvaan 11.

LA_So	Linear acceleration sensitivity(2)	FS = ±2		0.061		mg/LSB
		FS = ±4		0.122		
		FS = ±8		0.244		
		FS = ±16		0.488		
G_So	Angular rate sensitivity(2)	FS = ±125		4.375		mdps/LSB
		FS = ±250		8.75		
		FS = ±500		17.50		
		FS = ±1000		35		
		FS = ±2000		70		

Kuva 11. LSM6DSM-liikeanturin herkkyysasteet (STMicroelectronics 2017).

Kyseisiä herkkyysasteita tarvitaan, jotta saadaan kiihtyvyys- ja kulmanopeusanturin mittaama raakadata muutettua oikeaan muotoon. Kiihtyvyysanturin mittaama raakadata muutetaan seuraavalla menetelmällä G-voimiksi g.

Työssä käytetty kiihtyvyysanturin mittaama taso on $\pm 2\ g$, eli kyseinen taso on siis yhteensä $4\ g$. $4\ g$ voidaan esittää myös muodossa $4000\ \mathrm{m}g$ (milligramma-voima). Kiihtyvyysanturin mittaaman akselin raakadatan pituus on 16 bittiä, mikä vastaa kokonaislukua 65535, kun kaikki 16 bittiä ovat arvoltaan ykkösiä. Kiihtyvyysanturin mitatessa siis kiihtyvyyttä tietyn akselin osalta $\pm 2\ g$ tasossa, mahtuu kyseiseen tasoon 65535 erilaista mittaustulosta. (Williams 2017.) Tästä voidaan johtaa kiihtyvyysanturin mittaaman akselin herkkyystaso:

$$\frac{4000\ \mathrm{m}g}{65535} = 0{,}061\ \mathrm{m}g \qquad (3)$$

Kyseinen herkkyystaso $0{,}061\ \mathrm{m}g/LSB$ esiintyy myös kuvassa 11. Vähiten merkitsevän bitin muuttuessa siis yhdellä, muuttuu mitatun akselin kiihtyvyys $0{,}061\ \mathrm{m}g$:lla. Käyttämällä seuraavaa kaavaa voidaan mitatun kiihtyvyysanturin akselin raakadata (X) esittää G-voimissa:

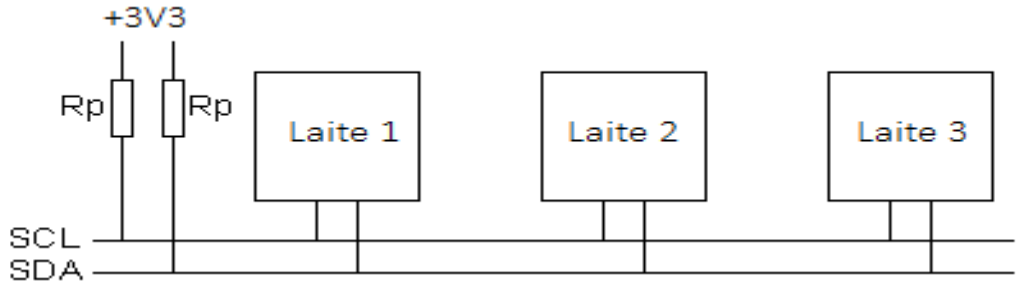
$$\frac{X \times 0{,}061\ \mathrm{m}g}{1000\ \mathrm{m}g} = X \qquad (4)$$

Kulmanopeusanturin herkkyystason määrittäminen perustuu samaan ideaan, kuin kiihtyvyysanturinkin. Käyttämällä seuraavaa kaavaa voidaan mitatun kulmanopeusanturin akselin raakadata (Y) esittää dps-yksiköinä:

$$\frac{Y \times 70\ \mathrm{mdps}}{1000\ \mathrm{mdps}} = Y \qquad (5)$$

3 I²C-VÄYLÄ

I²C-väylä on sarjamuotoinen, kaksisuuntainen tiedonsiirtoväylä. Sarjaväylän on kehittänyt Philips Semiconductor (nykyisin NXP Semiconductors) vuonna 1982. (Hutasu 2017.) I²C-väylä on erittäin suosittu sen helppokäyttöisyyden vuoksi. Väylä rakentuu kahdesta linjasta, data- (SDA) ja kellolinja (SCL). Kellolinjaa käytetään synkronoimaan lähetettävä data, joka kulkee I²C-väylässä. Datalinjaa käytetään datan siirtämiseen. Toimiakseen tulee väylän data- ja kellolinja kytkeä positiiviseen käyttöjännitteeseen (Kuva 12) ylösvetovastuksilla R_p.



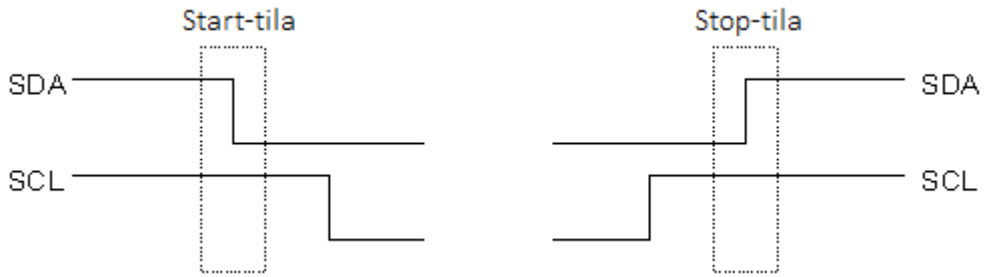
Kuva 12. I²C-väylän topologia (Hutasu 2017).

Väylään liitetyt laitteet ovat joko isäntä- tai orjalaitteita (master tai slave). Isäntälaite on väylää hallitseva laite, joka ohjaa data- ja kellolinjaa. Orjalaitteet ovat laitteita, jotka vastaavat isäntälaitteelle. Ainoastaan isäntälaite voi aloittaa datan siirtämisen väylää pitkin. Jokainen väylään liitetty laite voi toimia lähettäjänä tai vastaanottajana. Lähettäjäksi kutsutaan laitetta, joka lähettää väylään dataa ja vastaavasti dataa vastaanottavaa, vastaanottajaksi (STMicroelectronics 2017). I²C-väylässä voi olla ja yleensä onkin useampi orjalaite, mutta yleensä vain yksi isäntälaite. Väylään on kuitenkin mahdollista liittää useampi isäntälaite, mutta kyseinen menetelmä on epätavallinen. Väylässä on kuitenkin aina vähintään yksi isäntä- ja orjalaite. Väylään yhdistetyt laitteet erotellaan toisistaan osoitteiden avulla. Jokaisella laitteella on oma uniikki osoitteensa. Laitteen osoite löytyy laitteen valmistajan datalehdestä. Laitteiden osoitteet ovat pituudeltaan 7 tai 10 bitin pituisia (Robot Electronics 2018).

Isäntälaitteen kirjoittaessa väylään osoite, tulee orjalaitteen vastata ACK-bitillä (acknowledge), mikäli osoite vastaa orjalaitteen osoitetta. ACK-bitillä vastaaminen aiheuttaa

SDA-linjan menemisen nollaan. Mikäli isäntälaitteen kirjoittama osoite on väärin tai osoitetta ei ole kirjoitettu lainkaan, SDA-linjassa ei tapahdu muutosta ja ACK-bitti tulkitaan NACK-bitiksi (not acknowledge). (Hutasu 2017.)

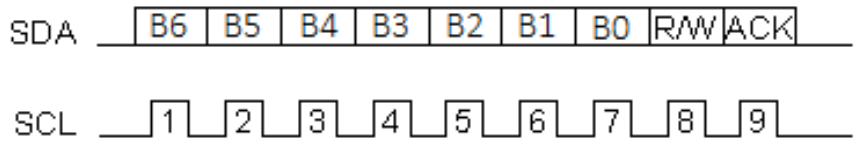
Kaikki tiedonsiirto I²C-väylässä alkaa aina START-tilalla ja päättyy STOP-tilaan. Isäntälaite luo väylään START-tilan, jonka jälkeen orjalaitteet alkavat välittömästi kuunnella väylää. Isäntälaite tekee myös väylään STOP-tilan, kun tiedonsiirto on kokonaisuudessaan valmis. START ja STOP-tilat ovat ainoat tilat (Kuva 13), jolloin sallitaan muuttaa SDA-linjaa SCL-linjan olleessa ylhäällä (Hutasu 2017).



Kuva 13. START ja STOP-tilat (Hutasu 2017).

Dataa siirretään I²C-väylässä 8 bitin ketjuissa. 8 bitin ketju lähetetään SDA-linjaa pitkin alkaen eniten merkitsevästä bitistä (MSB). (Robot Electronics 2018.)

START-tilan jälkeen, isäntälaite luo väylään 8 bitin ketjun vaatiman kahdeksan pulssin pituisen kellojakson jota seuraa yhdeksäs pulssi, minkä aikana orjalaite vastaa vastasiko lähetetty osoite hänen osoitettaan ACK-bitillä. Ensimmäiset 7 bittiä sisältävät laitteen osoitteen ja viimeinen 8 bitti kertoo, onko kyseessä luku- vai kirjoitusoperaatio (R/W). Kahdeksannen bitin arvon ollessa 1 on kyseessä lukuoperaatio ja vastaavasti bitin arvon ollessa 0 kirjoitusoperaatio (STMicroelectronics 2017). Mikäli isäntälaite lukee dataa orjalaitteelta, vastaa isäntälaite yhdeksännen kellopulssin aikana ACK-bitillä luettuun dataan. Kuvassa 14 on esitetty START-tilan jälkeiset kellopulssit ja bitit. Loput bitit jotka lähetetään I²C-väylään, olivat ne sitten kirjoitettavia tai luettavia bittejä kellotetaan myös väylälle samalla tavalla, 8 bitin ketjussa + ACK/NACK-bitin vaatima kellopulssi (STMicroelectronics 2017). Kirjoitus- ja lukuoperaatioiden jälkeen tehdään väylään STOP-tila isäntälaitteen toimesta, jolloin orjalaitteet lopettavat väylän kuuntelemisen.



Kuva 14. START-tilan jälkeinen toiminta (Hutasu 2017).

3.1 LSM6DSM-liikeanturin ja LPC11U14FBD48-mikrokontrollerin välinen kommunikointi

Liikeanturin ja mikrokontrollerin välinen kommunikointi toteutettiin työssä käyttäen I²C-väylää. Kommunikointi I²C-väylää käyttäen vaati kolmea johdinta SDA, SCL ja SA0. Kaikki edellä mainitut johtimet ovat kytketty suoraan mikrokontrollerilta liikeanturille. Lisäksi SDA- ja SCL-linjat ovat kytketty 10 kΩ ylösvetovastuksilla +3,3 V:n positiiviseen käyttöjännitteeseen. SDA- ja SCL-linjoja käytetään tiedon siirtämiseen I²C-väylää pitkin. SA0-pinnin avulla voidaan muuttaa liikeanturin osoitteen viimeistä bittiä.

LSM6DSM-liikeanturin osoitteen viimeinen bitti saa arvon 1, jos SA0-pinni on liitetty +3,3 V:n positiiviseen käyttöjännitteeseen, näin saaden osoitteeksi kokonaisuudessaan 1101011. Vastaavasti jos SA0-pinni on kytketty maahan, muuttuu viimeinen bitti nollaksi ja osoite on 1101010. (STMicroelectronics 2017.) SA0-pinni mahdollistaakin siis kahden identtisen laitteen kytkemisen samaan I²C-väylään. Tässä työssä SA0-pinni kytkettiin nollatasoon.

Kommunikointi väylässä aloitetaan START-tilan luomisella, jonka isäntälaite eli LPC11U14FBD48-mikrokontrolleri luo. Seuraavaksi mikrokontrolleri kirjoittaa väylään 8 bitin mittaisen ketjun, josta 7 ensimmäistä bittiä sisältävät LSM6DSM-liikeanturin osoitteen eli 1101010 ja viimeinen kahdeksas bitti (R/W) kertoo, lukeeko mikrokontrolleri dataa liikeanturilta vai kirjoitetaanko dataa liikeanturille. Kirjoitusoperaatiota tehtäessä liikeanturille tulee kahdeksannen bitin arvon olla 0, näin ollen väylään kirjoitettava osoite on kokonaisuudessaan 11010100. Lukuoperaatiota tehtäessä kahdeksannen bitin arvo on 1, joten väylään kirjoitettava osoite on 11010101. Osoitteen kirjoittamisen jälkeen liikeanturi vertaa omaa osoitettaan väylään kirjoitetun osoitteen 7 ensimmäiseen bittiin. Mikäli kirjoitettu osoite vastaa liikeanturin osoitetta, vastaa liikeanturi ACK-bitillä väylään.

Mikrokontrollerin luettua ACK-bitti, kirjoitetaan väylälle seuraavaksi 8-bittinen osoiterekisteri. Kirjoitettu osoiterekisteri kertoo mihin liikeanturin rekisteriin kirjoitus- tai lukuoperaatio kohdistuu. Liikeanturi sisältää rekistereitä, jotka ohjaavat liikeanturin toimintaa ja joista voidaan lukea kiihtyvyys- ja kulmanopeusarvoja sekä lämpötila. Liikeanturin rekisterit ovat pituudeltaan 8-bittisiä. Lähetettyyn osoiterekisteriin liikeanturi vastaa väylään ACK-bitillä. Mikrokontrollerin luettua liikeanturin lähettämä ACK-bitti, suorittaa se seuraavaksi luku- tai kirjoitusoperaation kyseiseen rekisteriin, riippuen aikaisemmin lähetetyn osoitteen kahdeksannen bitin arvosta (R/W). Kirjoitusoperaatioon liikeanturi vastaa väylään ACK-bitillä, luettuaan mikrokontrollerin lähettämän datan. Lukuoperaatioon mikrokontrolleri vastaa väylään ACK-bitillä, luettuaan liikeanturin lähettämän datan. Lähetettävä tai luettava data eli bitit ovat pituudeltaan 8-bittisiä. I²C-väylässä siirrettyjen bittien määrä on rajoittamaton, eli samaan rekisteriin voidaan kirjoittaa tai rekisteristä/rekistereistä voidaan lukea useaan kertaan saman lähetyksen aikana (STMicroelectronics 2017). Kirjoitus- ja lukuoperaatioiden jälkeen tiedonsiirto lopetetaan aina STOP-tilaan, jonka mikrokontrolleri luo.

3.1.1 Liikeanturin rekisterin lukeminen

Yksittäisen LSM6DSM-liikeanturin rekisterin lukeminen I²C-väylässä toimii seuraavalla tavalla. Aluksi LPC11U14FBD48-mikrokontrolleri tekee väylään START-tilan, jota seuraa LSM6DSM-liikeanturin osoitteen kirjoittaminen väylään, johon merkitään R/W-bitillä että seuraava operaatio on kirjoitusoperaatio. Kirjoitettava osoite väylään on siis muodossa 11010100. Liikeanturi kuittaa osoitteen ACK-bitillä, mikäli osoite vastaa liikeanturin osoitetta. ACK-bitin luettuaan mikrokontrolleri kirjoittaa väylään liikeanturin osoiterekisterin, josta dataa tullaan lukemaan. Liikeanturi kuittaa jälleen väylään kirjoitetun osoitteen ACK-bitillä. ACK-bitin luettuaan mikrokontrolleri tekee väylään uuden START-tilan (repeated start, SR). SR-tilan jälkeen mikrokontrolleri kirjoittaa väylään liikeanturin osoitteen + osoittaa R/W-bitillä, että seuraava operaatio on lukuoperaatio. Kirjoitettava osoite on siis muodossa 11010101. Liikeanturi kuittaa ACK-bitillä, jos kirjoitettu osoite vastaa liikeanturin osoitetta. Liikeanturi jatkaa tämän jälkeen lähetystä kirjoittamalla väylään rekisterin sisältämän datan. Datan luettuaan mikrokontrolleri kirjoittaa väylään ACK-bitin, jota seuraa STOP-tilan tekeminen. Kuvassa 15 on esitetty liikeanturin ja mikrokontrollerin välinen kommunikointi I²C-väylässä lukuoperaatiota suorittaessa.

Table 17. Transfer when master is receiving (reading) one byte of data from slave

Master	ST	SAD + W		SUB		SR	SAD + R			NMAK	SP
Slave			SAK		SAK			SAK	DATA		

Kuva 15. Lukuoperaatio I²C-väylässä liikeanturin rekisteristä (STMicroelectronics 2017).

3.1.2 Liikeanturin rekisteriin kirjoittaminen

LSM6DSM-liikeanturin rekisteriin kirjoittaminen I²C-väylässä toimii samalla periaatteella kuin lukuoperaatiota suorittaessa muutamaa poikkeusta lukuunottamatta. Väylässä kommunikointi aloitetaan LPC11U14FBD48-mikrokontrollerin tekemällä START-tilalla, jota seuraa liikeanturin osoite + R/W bitti. Koska kyseessä on kirjoitusoperaatio, on väylään kirjoitettava osoite 11010100. Liikeanturi kuittaa osoitteen ACK-bitillä, tarkastettuaan että väylään kirjoitettu osoite vastaa omaa osoitettaan. Mikrokontrolleri jatkaa kommunikointia luettuaan ACK-bitin, kirjoittamalla väylään liikeanturin osoiterekisterin johon kirjoitusoperaation tullaan kohdistamaan. Liikeanturi kuittaa ACK-bitillä väylään kirjoitetun osoitteen. ACK-bitin luettuaan mikrokontrolleri kirjoittaa väylään datan, jonka liikeanturi kuittaa ACK-bitillä luettuaan sen. Mikrokontrollerin luettua ACK-bitti tehdään väylään STOP-tila mikrokontrollerin toimesta, joka päättää väylässä kommunikoinnin. Kuvassa 16 on esitetty liikeanturin rekisteriin kirjoittaminen I²C-väylässä.

Table 15. Transfer when master is writing one byte to slave

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Kuva 16. Kirjoitusoperaatio I²C-väylässä liikeanturin rekisteriin (STMicroelectronics 2017).

4 KEHITYSYMPÄRISTÖ

4.1 LPCXpresso

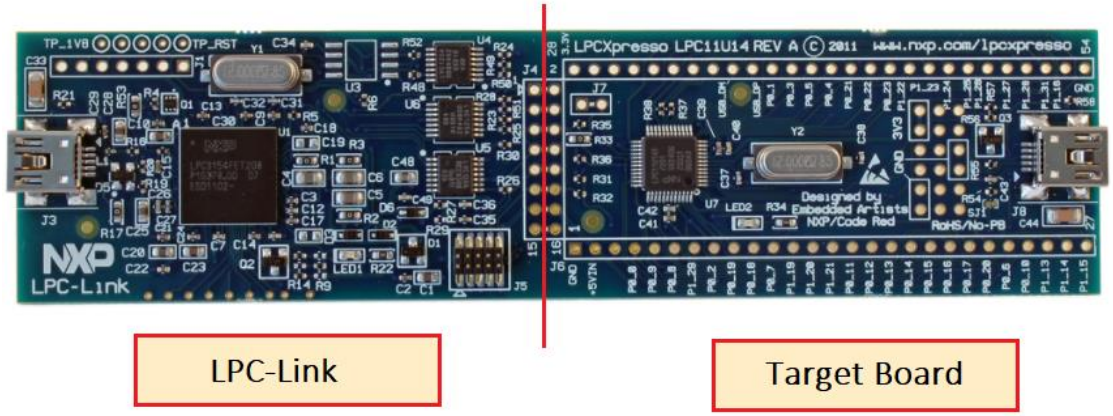
LPCXpresso on NXP Semiconductors:in kehittämä ohjelmointiympäristö. Ohjelmointiympäristö on luotu Eclipse nimisen ohjelmointiympäristön pohjalta. LPCXpresso tarjoaa ohjelmistokehittäjille tehokkaan ja edullisen ympäristön kehittää ja testata sulautettuja ohjelmistoja. Ohjelmointiympäristö on tarkoitettu NXP Semiconductors:in valmistamiin ARM mikroprosessiarkkitehtuuriin pohjautuvien LPC-mikrokontrollerien kehitysympäristöksi. (NXP Semiconductors 2013.)

Työssä käytetyn LPC11U14FBD48-mikroprosessorin sulautettu ohjelmisto kehitettiin käyttämällä kyseistä ohjelmointiympäristöä.

4.2 LPCXpresso-alusta

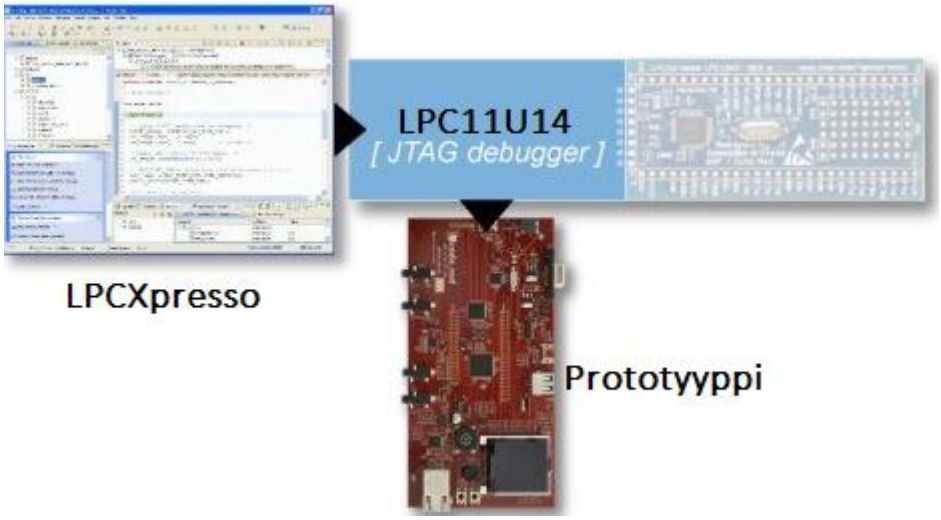
LPCXpresso-alusta on NXP Semiconductors ja Embedded Artists yrityksien tekemä kehitysalusta. Kehitysalusta rakentuu target ja LPC-link osista. Target osa sisältää ARM mikroprosessiarkkitehtuuriin perustuvan LPC-mikrokontrollerin ja liitäntöjä ulkoisten laitteiden liittämiseksi alustaan. LPC-link on debuggeri, jota käytetään alustaan liitetyn ulkoisen laitteen testaamiseen ja ohjelmointiin. LPC-link tukee JTAG- ja SWD-rajapintoja. Kehitysalustasta löytyy mini-USB liitäntä, jonka avulla alustaa voidaan ohjelmoida yhdistämällä alusta tietokoneen USB-porttiin. (NXP Semiconductors 2013.) Kuvassa 17 on havainnollistettu kehitysalustan rakenne.

Kehitysalustan target-osaa voidaan käyttää itsessään jo erilaisten sovellusten kehittämiseen alustan monipuolisten ominaisuuksien pohjalta. Tässä työssä kehitetty prototyyppi oli valmistettu erillisenä laitteena, jonka ohjelmoimiseen ja testaamiseen käytettiin alustan sisältämää LPC-link debuggeria.



Kuva 17. LPCXpresso-alusta (NXP Semiconductors 2018).

Työn sulautetun ohjelmiston kehittämisessä käytettiin LPCXpresso-ohjelmointiympäristöä, joka tukee työssä käytettyä LPC11U14-alustaa. Alustan ansioista työssä voitiin kehittää ja testauttaa sulautettu ohjelmisto prototyypin LPC11U14FBD48-mikroprosessorille, käyttämällä alustan sisältämää JTAG-rajapintaa. Kuvassa 18 on esitetty prototyypin kehitysympäristö.



Kuva 18. Prototyypin kehitysympäristö (LPC Tools 2014).

4.3 C-kieli

C-ohjelmointikieli on 1970-luvun alussa kehitetty ohjelmointikieli, joka alun perin suunniteltiin Unix-käyttöjärjestelmille. Kielen on alun perin kehittänyt Dennis Ritchie. C-kieli on yhä tänä päivänäkin yksi laajimmin käytetyistä ohjelmointikielistä. (Bell-Labs 2003.)

C-kielestä niin käytetyn ohjelmointikielen tekevät sen tehokkuus, yksinkertaiset komennot ja siirrettävyys. Suurin osa C-kielessä käytetyistä käskyistä tulee suoraan englannin kielestä. C-kielessä yhdistyvät laiteläheisen ja korkean ohjelmointikielen tason ominaisuudet. C-kielen tärkeimpiin ominaisuuksiin kuuluu C-kielen kääntäjän löytyminen melkein jokaisesta sulautetusta laitteistosta, vähäinen muistinkulutus, suorituskyky sekä matalan tason pääsy laitteiston muistiin, I/O-portteihin jne. Kyseisten ominaisuuksien pohjalta suurin osa sulautetuista ohjelmistoista toteutetaankin käyttämällä laiteläheistä C-kieltä, kuten tämän työn sulautettu ohjelmisto.

5 PROTOTYYPIN SUUNNITTELU JA TOTEUTUS

5.1 Vaatimusmäärittely

Työn tavoitteeksi määriteltiin suunnitella laitteisto, joka indikoi käyttäjälle liike- ja kulmanopeusanturilta luettavia arvoja sekä näiden tuloksia. Laitteiston tulisi indikoida käyttäjälle ainoastaan laitteeseen kohdistuvaa negatiivista kiihtyvyyttä eli negatiivisen nopeuden jatkuvaa kasvamista tai yksinkertaistettuna laitteen hidastuvuutta.

Työn mittaustavoite oli tutkia jatkuvaa negatiivista kiihtyvyyttä väliltä $-0{,}01\ g\ -\ -0{,}23\ g$ eli $-0{,}098\ \mathrm{m/s^2}\ -\ -2{,}254\ \mathrm{m/s^2}$. Laitteeseen kohdistuvia äkillisiä kiihtyvyyden muutoksia ei tule indikoida, vaikka mitattu kiihtyvyys täyttäisi mittaustavoitteen. Kulmanopeusanturille ei määritelty mittaustavoitetta, koska anturia käytettiin ainoastaan tunnistamaan, onko laitteella kulmanopeutta.

Laitteeseen kohdistuu myös kulmakiihtyvyyttä laitteen kääntyessä, joka voidaan havaita muutoksena laitteen kulmanopeudessa. Kyseistä kulmanopeudesta johtuvaa kiihtyvyyttä ei tule indikoida käyttäjälle.

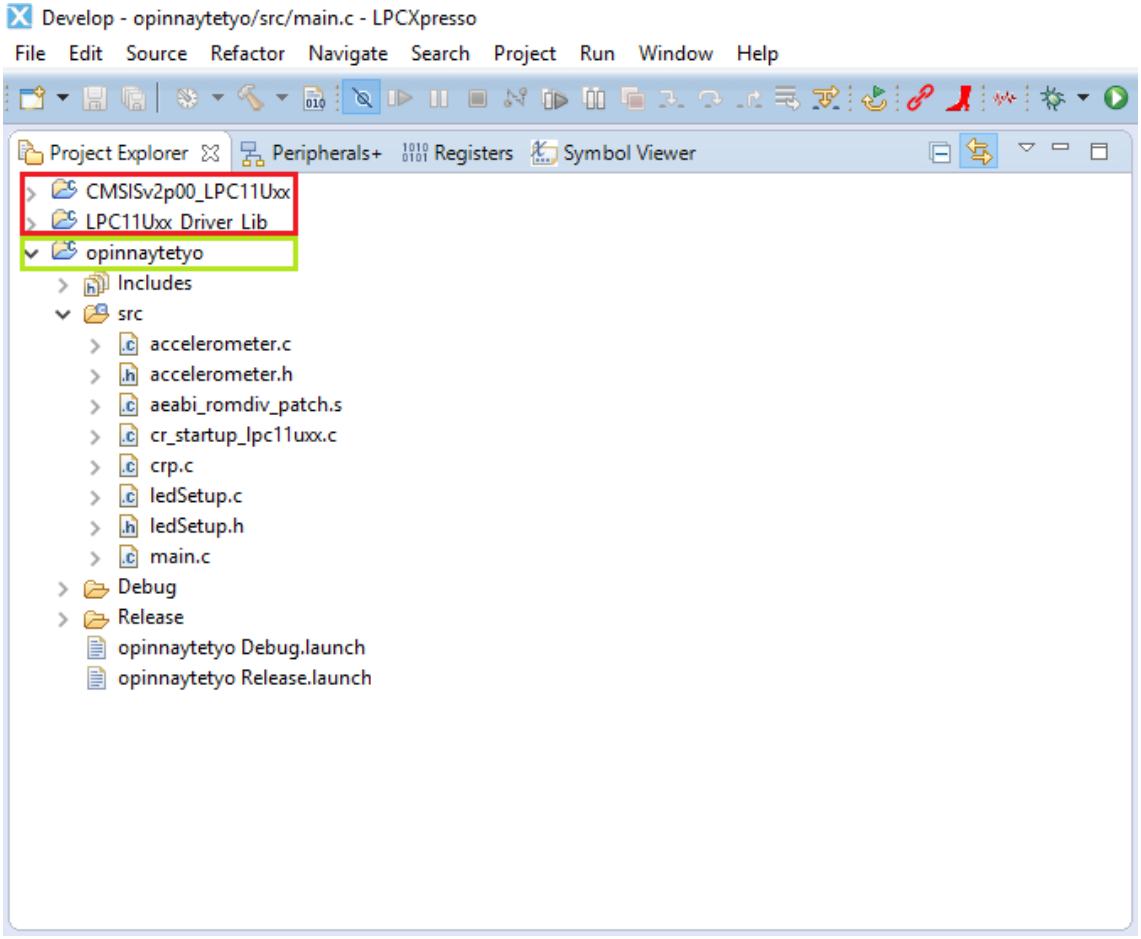
Työn vaatimusmäärittelyn mukainen laite saavutettaisiin toteuttamalla sulautettu ohjelmisto mikrokontrollerille, jonka avulla voitaisiin lukea liikeanturilta saatavia arvoja ja indikoida niistä johdettuja tuloksia käyttäjälle I/O-portteihin kytkettyjen ledien avulla.

5.2 Projektin luominen

Työn sulautettu ohjelmisto toteutettiin käyttämällä LPCXpresso-ohjelmointiympäristöä. Sulautetun ohjelmiston kehittäminen työssä käytettyyn LPC11U14FBD48-mikrokontrolleriin vaati ohjelmistokirjastoa (kuvassa 19 CMSISv2p00_LPC11Uxx), joka sisältää mikrokontrollerin toiminnan kannalta välttämättömät ajurit ja väliohjelmiston. Lisäksi toinen ohjelmistokirjasto (kuvassa 19 LPC11Uxx_Driver_Lib) vaadittiin LPC11U14-alustalle, joka mahdollisti sulautetun ohjelmiston testaamisen ja kirjoittamisen laitteiston mikrokontrollerille, alustan sisältämän JTAG-rajapinnan kautta. Ohjelmistokirjastot ovat valmiiksi saatavilla NXP Semiconductors:in toimesta. Tässä säästettiinkin huomattavasti

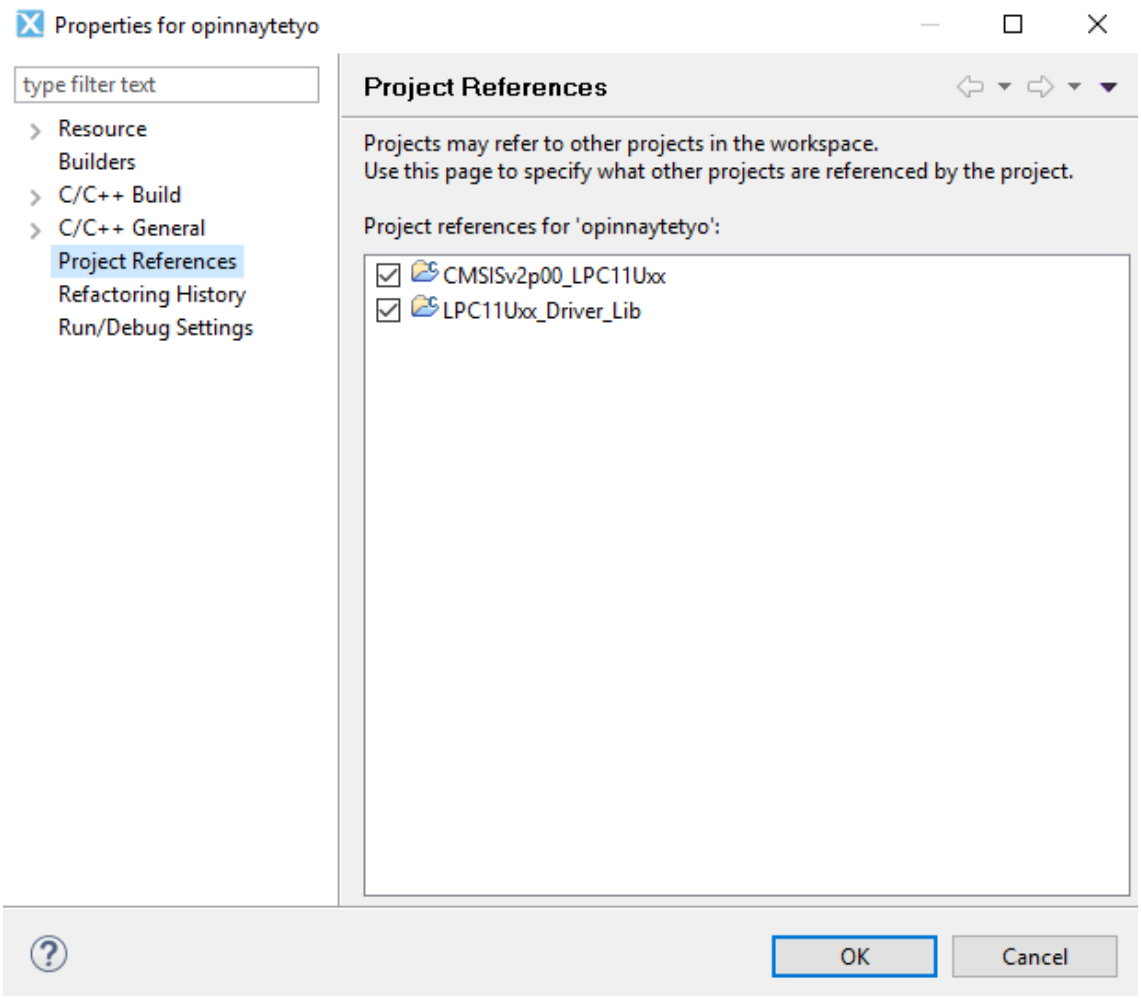
aikaa työn toteuttamisen kannalta, koska välttämättömiä ajureita ja väliohjelmistoa ei tarvinnut itse implementoida.

Työssä vaaditut ohjelmistokirjastot on merkitty kuvaan 19 punaisella kehyksellä ja vihreällä toteuttamani projekti, joka sisältää laitteiston sulautetun ohjelmiston.



Kuva 19. Ohjelmointiympäristön työtila.

Ohjelmistokirjastojen sisältämien tiedostojen käyttämistä varten tuli projektit linkittää toisiinsa. Projekti voidaan linkittää toisiin samassa työtilassa oleviin projekteihin, projektin ominaisuuksia muokkaamalla. Kuvassa 20 on esitetty opinnaytetyo-projektin linkittäminen LPC11U14FBD48-mikrokontrollerin ja LPC11U14-alustan ohjelmistokirjastoihin.



Kuva 20. Projektin linkittäminen ohjelmistokirjastoihin.

5.3 Kytkentä

Tässä luvussa käydään läpi, miten LPC11U14FBD48-mikrokontrolleri, LSM6DSM-liikeanturi ja tuloksien indikointiin käytetyt ledit ovat elektronisesti kytketty. Lisäksi luvussa käydään läpi, miten kyseiset komponentit on kytketty toisiinsa näin muodostaen yhtenäisen laitteiston.

5.3.1 Mikrokontrolleri

Mikrokontrolleri on työn laitteiston äly, joka ohjaa mikrokontrolleriin liitettyjä ledejä ja LSM6DSM-liikeanturia. Kuvassa 21 on esitetty LPC11U14FBD48-mikrokontrollerin kaikki 48 pinniä. Osalla mikrokontrollerin pinneillä on erilaisia toiminnallisuuksia. Esimerkiksi pinniä numero 16 voidaan käyttää yleisenä I/O-porttina tai SDA-linjana I²C-väylässä. Pinnin toiminnallisuuden määrittäminen tapahtuu muokkaamalla kyseisen pinnin IOCON-rekisteriä.

Mikrokontrollerin ohjelmointia ja debuggaamista varten tehtiin JTAG-kaapeli, jonka avulla laite voitiin yhdistää LPC11U14-alustan JTAG-rajapintaan. Laitteistoon rakennettiin JTAG-liitin, johon kytkettiin mikrokontrollerin ohjelmointia ja debuggaamista varten tarvittavat pinnit. Kuvassa 22 on esitetty LPC11U14-alustan JTAG-rajapinnan kytkentäkaavio. LSM6DSM-liikeanturi liitettiin mikrokontrolleriin käyttämällä I²C-väylä topologiaa. I²C-väylän muodostamiseen, yhdistettiin liikeanturi ja mikrokontrolleri kolmella johtimella toisiinsa. Työssä käytettiin yhteensä 24 lediä liikeanturilta luettavien tuloksien indikoimiseen. Ledit kytkettiin mikrokontrollerin eri I/O-portteihin mahdollistaen ledien päälle ja pois kytkemisen. Ledit jaettiin kahteen eri ryhmään L ja R, jossa molemmissa ryhmissä oli 12 lediä. Esimerkkinä LED 10R, on R-ryhmän kymmenes LED. Taulukossa 1 on esitetty mikrokontrollerin pinnien kytkennät.

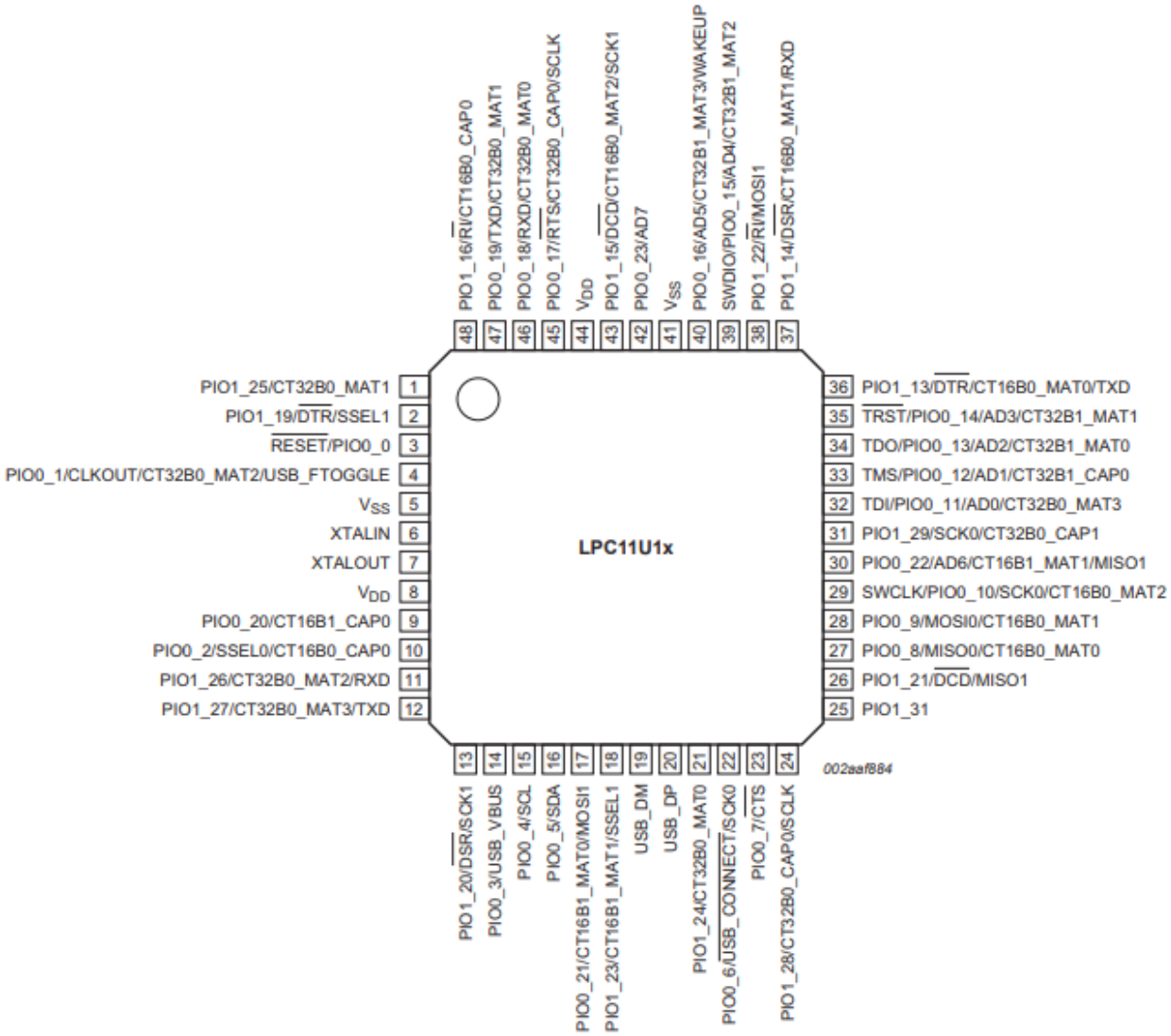
Taulukko 1. LPC11U14FBD48-mikrokontrollerin pinnien kytkennät.

Pinnin numero	**Nimi**	**Kuvaus**
1	PIO1_25	LED 12R
2, 4, 9, 10, 11, 12, 13, 14, 18, 19, 20	-	Ei kytketä
3	RESET	JTAG_RESET
5, 41	VSS	GND
6,7	XTALIN, XTALOUT	10pF kondensaattorin kautta GND
8, 44	VDD	+3,3V
15	SCL	Liikeanturin SCL

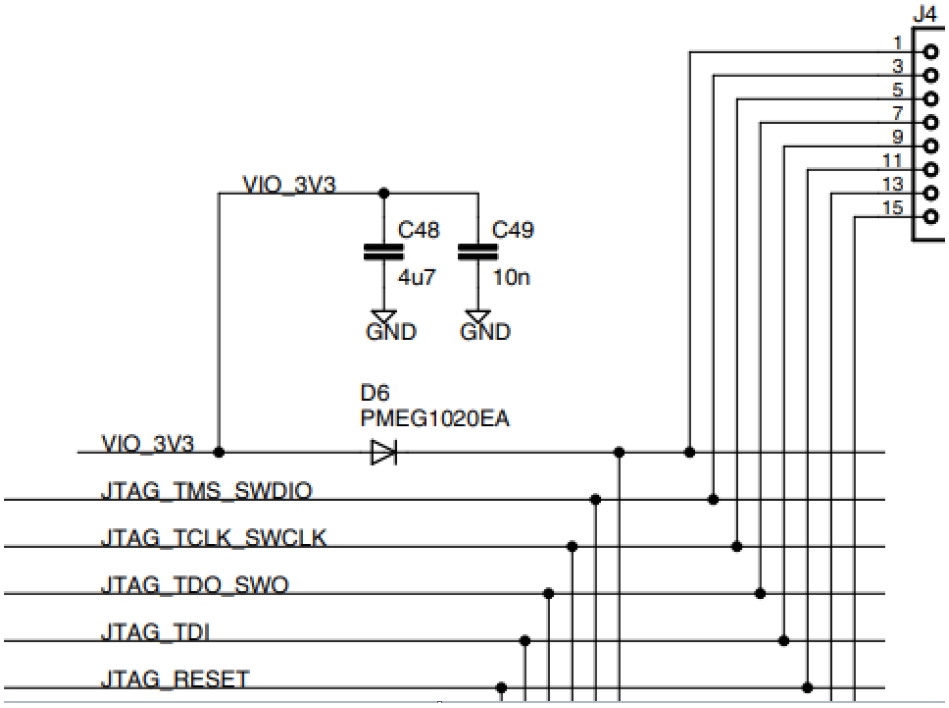
(Jatkuu)

Taulukko 1 (jatkuu).

Pinnin numero	Nimi	Kuvaus
16	SDA	Liikeanturin SDA
17	PIO0_21	Liikeanturin SA0
21	PIO1_24	LED 1L
22	PIO0_6	LED 2L
23	PIO0_7	LED 3L
24	PIO1_28	LED 4L
25	PIO1_31	LED 5L
26	PIO1_21	LED 6L
27	PIO0_8	LED 7L
28	PIO0_9	JTAG_TDO_SWO
29	SWCLK	JTAG_TCLK_SWCLK
30	PIO0_22	LED 8L
31	PIO1_29	LED 9L
32	PIO0_11	LED 10L
33	PIO0_12	LED 11L
34	PIO0_13	LED 12L
35	PIO0_14	LED 1R
36	PIO1_13	LED 2R
37	PIO1_14	LED 3R
38	PIO1_22	LED 4R
39	SWDIO	JTAG_TMS_SWDIO
40	PIO0_16	LED 5R
42	PIO0_23	LED 6R
43	PIO1_15	LED 7R
45	PIO0_17	LED 8R
46	PIO0_18	LED 9R
47	PIO0_19	LED 10R
48	PIO1_16	LED 11R



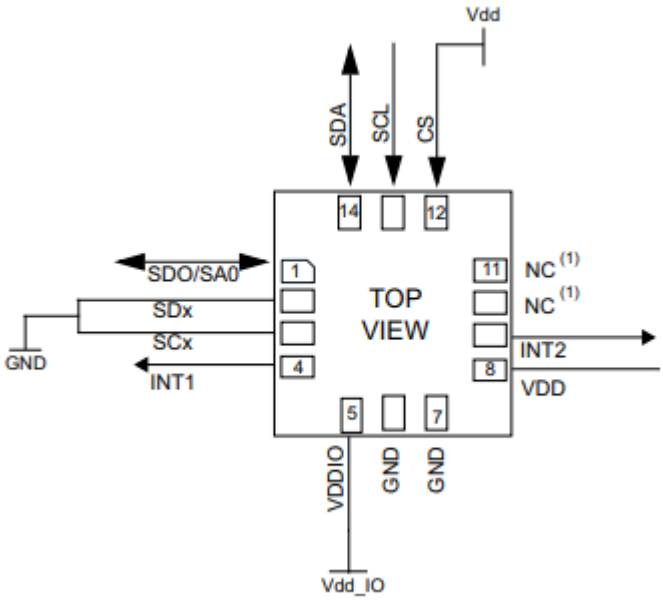
Kuva 21. LPC11U14FBD48-mikrokontrollerin pinnien konfiguraatio (NXP Semiconductors 2014).



Kuva 22. LPC11U14 alustan JTAG-rajapinta (Embedded Artists AB 2012).

5.3.2 Liikeanturi

LSM6DSM-liikeanturi kytkettiin Kuvan 23 mukaisesti. Liikeanturin pinnit SA0, SDA ja SCL kytkettiin taulukon 1 mukaan, LPC11U14FBD48-mikrokontrollerin SDA-, SCL- ja PIO0_21-pinneihin. Liikeanturin SDx- ja SCx-pinnit kytkettiin maahan. VDDIO- ja VDD-pinnit kytkettiin +3,3 V:n positiiviseen käyttöjännitteeseen. GND-pinnit kytkettiin nimensä mukaisesti maahan. CS-pinni kytkettiin +3,3 V:n positiivisen käyttöjännitteeseen, mikä mahdollisti kommunikoinnin I²C-väylässä. Loput liikeanturin pinnit jätettiin kytkemättä.

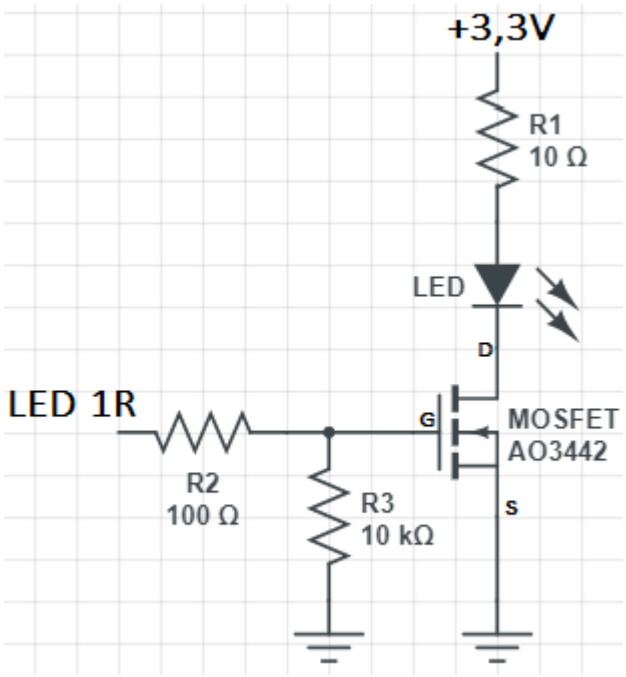


Kuva 23. LSM6DSM-liikeanturin kytkentä (STMicroelectronics 2017).

5.3.3 LED

Kaikki työssä käytetyt 24 lediä kytkettiin samalla periaatteella. Ledin anodi on kytketty +3,3 V:n positiiviseen käyttöjännitteeseen 10 Ω vastuksen kautta ja katodi MOSFETIN nielu (D) osaan. MOSFETIN lähde (S) kytkettiin maahan. Hila (G) on kytketty 100 Ω vastuksen läpi mikrokontrollerin pinneihin, taulukon 1 mukaisesti. Hila on kytketty myös 10 kΩ alasvetovastuksen kautta maahan, jotta mikrokontrolleri voi kytkeä varmasti ledin päälle tai pois päältä. Kuvassa 24 on esitetty ledien kytkentäkaavio.

Ledin päälle tai pois päältä kytkeminen tapahtui kirjoittamalla mikrokontrollerin I/O-porttiin positiivinen +3,3 V:n tai 0 V:n jännite. Hilalle tuotaessa +3,3 V:n positiivinen ohjausjännite, alkaa virta kulkemaan nielun ja lähteen välillä sytyttäen ledin. Vastaavasti jos hilalle ei tuoda positiivista jännitettä, on kanava sulkeutunut ja ledi ei syty.

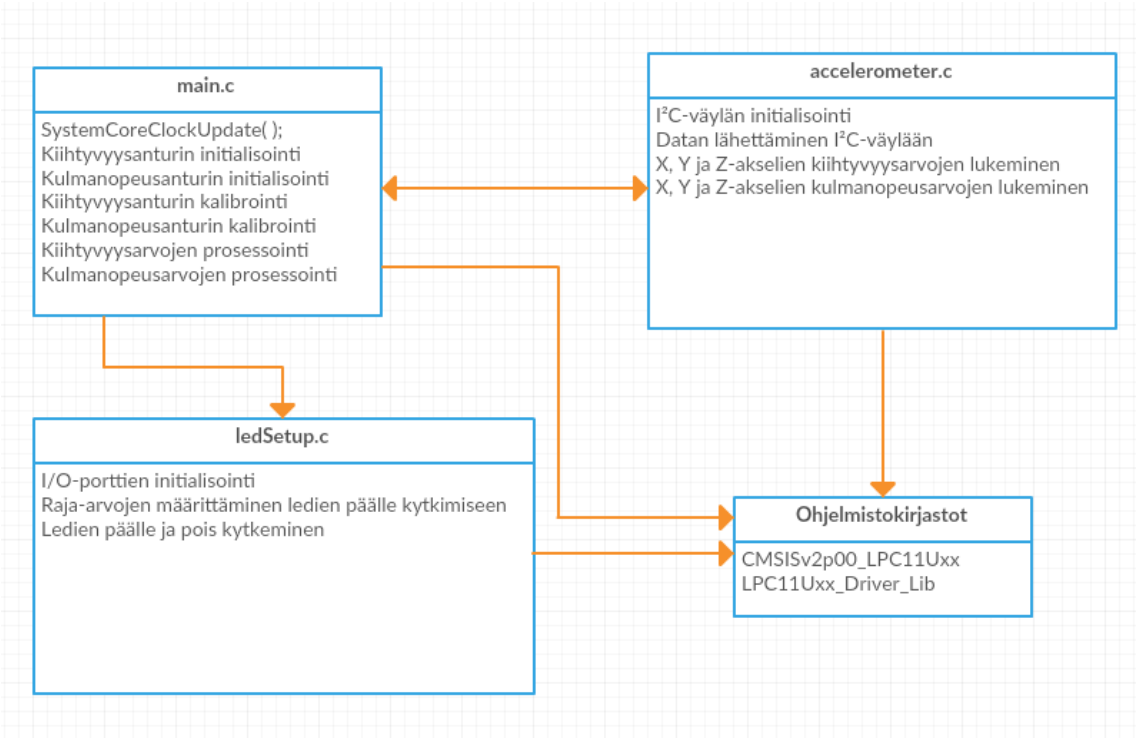


Kuva 24. LED kytkentäkaavio.

5.4 Sulautettu ohjelmisto

5.4.1 Arkkitehtuuri

Työn sulautettu ohjelmisto jaettiin useaan eri aliohjelmakokonaisuuteen. I/O-porttien initialisointia ja ledien ohjaamista varten tehtiin lähdekooditiedosto ledSetup.c ja otsikkotiedosto ledSetup.h. I²C-väylän initialisointia ja liikeanturilta arvojen lukemista varten tehtiin lähdekooditiedosto accelerometer.c ja otsikkotiedosto accelerometer.h. Main.c-lähdekooditiedosto sisältää sulautetun ohjelmiston funktionaalisen toiminnan ja josta kutsutaan aliohjelmien sisältämiä funktioita. Kyseisen ohjelmakokonaisuuden lisäksi, valmiina olevia ohjelmistokirjastojen sisältämiä aliohjelmia käytettiin sulautetun ohjelmiston kehittämisessä ja toteuttamisessa. Kuviossa 2 on esitetty sulautetun ohjelmiston arkkitehtuuri ja eri aliohjelmien funktionallisuus.



Kuvio 2. Sulautetun ohjelmiston arkkitehtuuri.

5.4.2 I²C-väylä

I²C-väylässä datan tiedonsiirtämistä varten, tulee väylä ensin initialisoida. I²C-väylän initialisointia varten tehtiin koodin 1 mukainen funktio accelerometer.c-lähdekooditiedostoon. I²C-väylä resetoidaan aluksi asettamalla PRESETCTRL-rekisterin ensimmäinen bitti arvoon 1, jota seurasi bitin asettaminen arvoon 0. SYSAHBCLKCTRL-rekisteri sisältää järjestelmän eri rajapintojen kellojen käyttöönottamisen. I²C-väylän kellon käyttöönottamista varten, asetetaan kyseisen rekisterin 5 bitti arvoon 1. PIO0_4-porttia käytettiin työssä SCL-linjana ja PIO0_5-porttia SDA-linjana. Kyseiset portit ovat toiminnallisuudeltaan oletuksena yleisiä I/O-portteja. Jotta kyseisiä portteja voitiin käyttää I²C-väylän kello- ja datalinjana, tuli portit aluksi resetoida ja tämän jälkeen asettaa funktionallinen toiminta vastaamaan I²C-väylän kello- ja datalinjaa muokkaamalla porttien rekistereitä. CONCLR-rekisterillä hallittaan CON-rekisteriä, joka hallitsee I²C-väylän toimintaa. CONCLR-rekisterin avulla voidaan resetoida I²C-väylän eri toiminnallisuuksien binäärimuuttujia. CONCLR-rekisterin sisältämien bittien arvon muuttaminen arvoon 1, aiheuttaa

CONSET-rekisterin binäärimuuttujien nollautumisen. Kyseisen rekisterin avulla resetoitiin I²C-väylän AAC-, SIC-, STAC- ja I2ENC-bitit. SCLL- ja SCLH-rekisterien arvojen määrittämisellä, konfiguroidaan I²C-väylän kellopulssin taajuus (NXP Semiconductors 2016). I²C-väylän kellopulssin taajuus määritellään seuraavalla kaavalla:

$$\mathrm{I^2C-kellopulssi} = \frac{\mathrm{I2CPCLK}}{\mathrm{SCLH + SCLL}} \quad (6)$$

I2CPCLK on LPC11U14FBD48-mikrokontrollerin kellon kellotaajuus. Työssä käytetyn mikrokontrollerin kellotaajuus oli 48 MHz. SCLH- ja SCLL-rekisterien arvoksi määriteltiin 240 eli näin kaavan 6 mukaan, I²C-väylän kellopulssin taajuudeksi asetettiin 100 kHz. Initialisoinnin lopuksi aktivoitiin I²C-väylä asettamalla CONSET-rekisterin kuudes bitti arvoon 1.

```
** Function name:       I2CInit
void I2CInit( void )
{
            LPC_SYSCON->PRESETCTRL    |= (0x1<<1);
            LPC_SYSCON->PRESETCTRL    |= (0x2);    //disable reset to I2C unit (sec 3.5.2)

            LPC_SYSCON->SYSAHBCLKCTRL |= (1<<5);
            LPC_IOCON->PIO0_4 &= ~0x3F;     //I2C I/O config
            LPC_IOCON->PIO0_4 |= 0x01;      //I2C SCL
            LPC_IOCON->PIO0_5 &= ~0x3F;
            LPC_IOCON->PIO0_5 |= 0x01;      //I2C SDA

            /*--- Clear flags ---*/
            LPC_I2C->CONCLR = I2CONCLR_AAC | I2CONCLR_SIC | I2CONCLR_STAC | I2CONCLR_I2ENC;

            LPC_I2C->SCLL   = 0xF0;
            LPC_I2C->SCLH   = 0xF0;

            LPC_I2C->CONSET |= (1<<6);   //put I2C unit in master transmit mode (sec 15.8.1 and 15.7.1)

}
```

Koodi 1. I²C-väylän initialisointi.

5.4.3 Kiihtyvyysarvojen lukeminen ja prosessointi

Kiihtyvyysanturin X, Y ja Z-akselien kiihtyvyysarvojen lukemista varten, tehtiin jokaiselle akselille oma funktio accelerometer.c-lähdekooditiedostoon. Jokaisen kiihtyvyysanturin akselin kiihtyvyysarvojen lukemisen proseduuri oli sama, ainoastaan osoiterekisterit ja funktion parametrin nimi muutettiin funktiokohtaisesti eri akselien osalta. Liitteessä 1 on esitetty kiihtyvyysanturin X-akselin kiihtyvyysarvojen lukeminen. Funktiota kutsuttiin main.c-tiedostosta, jonka parametrina oli taulukko, johon kiihtyvyysanturilta luettavat 8 vähiten merkitsevää (LSB) ja 8 eniten merkitsevää bittiä (MSB) tallennettiin, joka myöhemmin muutettiin 16-bittiseksi sanaksi.

I²C-väylässä kommunikointi aloitetaan tekemällä START-tila CONSET-rekisteriä hyödyntäen. Tämän jälkeen odotetaan, kunnes START-tila on tehty onnistuneesti väylään. START-tilan jälkeen, kirjoitetaan väylään LSM6DSM-liikeanturin osoite ja osoitetaan R/W-bitillä, että seuraava operaatio tulee olemaan kirjoitusoperaatio. Osoitteen kirjoittamisen jälkeen nollataan CONSET-rekisterin binäärimuuttujat CONCLR-rekisteriä käyttämällä, jotta voidaan tarkkailla CONSET-rekisteristä, vastaako liikeanturi lähetettyyn osoitteeseen ACK-bitillä. Vastauksen saatua nousee CONSET-rekisterin kolmas bitti arvoon 1, joka osoittaa, että I²C-väylän tila on muuttunut. Väylään kirjoitetaan seuraavaksi liikeanturin osoiterekisteri, joka sisältää akselin kiihtyvyysarvon vähiten merkitsevät bitit. Osoitteen kirjoittamista seuraa jälleen binäärimuuttujien nollaaminen, jotta väylältä voidaan lukea liikeanturin lähettämä ACK-bitti. Vastauksen saatua, tehdään väylään uusi START-tila (repeated start). Onnistuneen START-tilan tekemisen jälkeen, kirjoitetaan väylään liikeanturin osoite, ja osoitetaan R/W-bitillä että seuraava operaatio on lukuoperaatio. CONSET-rekisterin kolmas bitti nousee taas arvoon 1, kun liikeanturi vastaa lähetettyyn osoitteeseen ACK-bitillä. Vastauksen saatua, mikrokontrolleri nollaa jälleen SIC-binäärimuuttujan, jonka arvo muuttuu, kun liikeanturi lähettää väylään osoiterekisterin sisältämän datan. Väylään kirjoitettu data tallennetaan taulukon indeksin arvoon 1. Datan tallentamisen jälkeen tekee mikrokontrolleri väylään ACK-bitin, jota seuraa STOP-tilan tekeminen. Lähetyksen päättämisen jälkeen nollataan kaikki CONSET-rekisterin binäärimuuttujat CONCLR-rekisteriä käyttämällä. Akselin kiihtyvyysarvon 8 eniten merkitsevää bittiä luetaan liikeanturilta samalla periaatteella, ainoastaan liikeanturin osoiterekisterin ja taulukon indeksin arvon muuttuessa.

On välttämätöntä nollata CONSET-rekisterin binäärimuuttujat käyttämällä CONCLR-rekisteriä jokaisen operaation jälkeen, jotta voidaan havaita, kun I²C-väylän tilassa tapahtuu muutos.

Kiihtyvyysarvojen prosessointi raakadatasta G-voimiksi suoritetaan main.c-tiedostossa, kiihtyvyysarvojen lukemisen jälkeen. Kiihtyvyysarvojen lukemisen ehtona oli ensin tarkistaa liikeanturin tilarekisteristä, onko kiihtyvyysanturilta luettavissa uutta dataa. Lisäksi kiihtyvyysanturi konfigurointiin niin, että uusia kiihtyvyysarvoja ei päivitetä rekisteriin ennen kuin, rekisterin eniten ja vähiten merkitsevät bitit oli luettu kokonaisuudessaan. Raakadatan muuntaminen G-voimiksi, toteutetaan siirtämällä liikeanturilta luetut eniten merkitsevät bitit vasemmalle 8 bitin verran. Tämän jälkeen eniten ja vähiten merkitsevät bitit yhdistetään OR-operaatiolla, jotta saadaan yhtenäinen 16-bittinen arvo. Johdettua 16-bittistä arvoa sovelletaan operaation jälkeen kaavan 4 mukaan, jotta raakadata voidaan esittää G-voimissa.

Kiihtyvyyttä pitää käsitellä yhtenä kokonaisena vektorisuureena. Vektorisuure saadaan seuraavasta kaavasta:

$$A = \sqrt{X^2 + Y^2 + Z^2} \quad (7)$$

Laitteen ollessa paikallaan on kokonaiskiihtyvyyden suuruus $1\ g$, johtuen Maan vetovoiman staattisesta voimasta. Laitteeseen kohdistuvan kiihtyvyyden muutoksen seurauksena, kokonaiskiihtyvyyden suuruus muuttuu. Näin ollen kokonaiskiihtyvyys on Maan vetovoiman staattinen voima + laitteen todellinen kiihtyvyys. Kokonaiskiihtyvyyttä laskettiin silmukassa, jotta kokonaiskiihtyvyydelle voitiin laskea keskiarvoa. Keskiarvon laskemisen johdosta voitiin laitteeseen kohdistuvien äkillisten kiihtyvyyden muutoksien merkitystä pienentää laitteen todellisesta kokonaiskiihtyvyydestä. Koodissa 2 on esitetty kiihtyvyysarvojen prosessointi ja kokonaiskiihtyvyyden laskeminen. Keskiarvollisen kokonaiskiihtyvyyden saamiseksi tuli silmukan jälkeen jakaa saatu tulos silmukan kierroksien määrällä. Kokonaiskiihtyvyyden keskiarvosta vähennettiin tämän jälkeen Maan vetovoiman aiheuttama $1\ g$ suuruinen staattinen voima, jonka tuloksena voitiin tarkastella laitteeseen kohdistuvaa todellista kiihtyvyyttä. Maan vetovoiman aiheuttaman staattisen voiman vähentämisen jälkeen tuli saatu tulos kalibroida vielä liikeanturin kalibroinnista johdetulla tuloksella.

```
/*Tarkastetaan onko uutta dataa luettavissa, lukemalla kiihtyvyysanturin rekisteristä
* onko tietty bitti 1/0 */
accDataCheck = readStatusRegisterAcc(accDataCheck);
/*Luetaan kiihtyvyysanturilta dataa, jos uutta dataa on luettavissa*/
if(accDataCheck){
/*Silmukka jossa lasketaan kokonaiskiihtyvyydelle keskiarvoa*/
for(int i = 0; i < average; i++){

/*****************************************************************************************/

//Luetaan kiihtyvyys Y-akselilta
yAxisAccelerometer(yAxisAccelerometerData);
//Muutetaan MSB ja LSB 16bittiseksi arvoksi
yOut16bit_g = ((short)(yAxisAccelerometerData[0] << 8 | yAxisAccelerometerData[1]));
//Muutetaan arvo todelliseen arvoon eli g-voimiksi
//(+-2g) datalehden sivu 23, Taulukko 3)
yOut_g = ((float) yOut16bit_g * 0.000061);


/*****************************************************************************************/

//Luetaan kiihtyvyys Z-akselilta
zAxisAccelerometer(zAxisAccelerometerData);
//Muutetaan MSB ja LSB 16bittiseksi arvoksi
zOut16bit_g = ((short)(zAxisAccelerometerData[0] << 8 | zAxisAccelerometerData[1]));
//Muutetaan arvo todelliseen arvoon eli g-voimiksi
//(+-2g) datalehden sivu 23, Taulukko 3)
zOut_g = ((float) zOut16bit_g * 0.000061);

/*****************************************************************************************/

//Luetaan kiihtyvyys X-akselilta
xAxisAccelerometer(xAxisAccelerometerData);
//Muutetaan MSB ja LSB 16bittiseksi arvoksi
xOut16bit_g = ((short)(xAxisAccelerometerData[0] << 8 | xAxisAccelerometerData[1]));
//Muutetaan arvo todelliseen arvoon eli g-voimiksi
//(+-2g) datalehden sivu 23, Taulukko 3)
xOut_g = ((float) xOut16bit_g * 0.000061);

acceleration += sqrt((xOut_g * xOut_g) + (yOut_g * yOut_g) + (zOut_g * zOut_g));

}

/*Jaetaan kiihtyvyys silmukan kierrosten määrällä, jotta saadaan keskiarvo*/
acceleration /= average;
/*Vähennetään maanvetovoima vakio 1g*/
acceleration -= 1;

/*Jos kiihtyvyyden kalibrointi arvo on > 0 vähennetään se mitatusta arvosta
* muuten lisätään mitattuun arvoon*/
if(accelerationOffSet < 0.0){
acceleration += accelerationOffSet;
}
else{
acceleration -= accelerationOffSet;
}
```

Koodi 2. Kiihtyvyysarvojen laskeminen.

5.4.4 Kulmanopeusarvojen lukeminen ja prosessointi

Liikeanturin kulmanopeusarvojen lukemista varten hyödynnettiin samanlaista ratkaisua, mitä kiihtyvyysarvojen lukemisessa käytettiin. Lukemisen ainoana erona, olivat liikeanturin osoiterekisterin osoitteet. Kulmanopeusanturin X-akselin kulmanopeusarvojen lukeminen on esitetty liitteessä 1. Myös kulmanopeusarvojen lukemisen ehtona oli ensin tarkistaa liikeanturin tilarekisteristä, onko kulmanopeusanturilta luettavissa uutta dataa. Lisäksi uusia kulmanopeusarvoja ei päivitetty rekisteriin, ennen kuin rekisterin eniten ja vähiten merkitsevät bitit oli luettu kokonaisuudessaan. Kulmanopeusarvojen lukemisen jälkeen, suoritettiin main.c-tiedostossa koodin 3 mukainen prosessointi. Saatua 16-bittistä arvoa sovellettiin kaavan 5 mukaan, jotta raakadata voitiin esittää dps-yksiköinä. Akselien kulmanopeusarvot muutettiin liukuluvuista kokonaisluvuiksi, jotta kalibroinnista johdettuja tuloksia voitiin soveltaa mitattuihin arvoihin. Kulmanopeusanturille ei työssä asetettu mittaustavoitetta, vaan käytettiin ainoastaan laitteen kulmanopeuden tunnistamiseen, joten dps-yksiköitä voitiin käsitellä kokonaislukujen tarkkuudella.

```
/*Tarkastetaan onko uutta dataa luettavissa, lukemalla kulmanopeusanturin rekisteristä
* onko tietty bitti 1/0 */
gyroDataCheck = readStatusRegisterGyro(gyroDataCheck);
/*Luetaan dataa kulmanopeusanturilta dataa, jos uutta dataa on luettavissa*/
if(gyroDataCheck){
/*Luetaan kulmanopeus X-akselin osalta*/
xAxisGyroscope(xAxisGyroscopeData);
//Muutetaan MSB ja LSB 16bittiseksi arvoksi
xOut16bit_dps = ((short)(xAxisGyroscopeData[0] << 8 | xAxisGyroscopeData[1]));
//Muutetaan arvo todelliseen arvoon eli DPS (degrees per second)
//(+-2000dps) datalehden sivu 23, Taulukko 3)
xOut_dps = ((float) xOut16bit_dps * 0.07);
/*Vähennetään mitatusta arvosta kalibroinnin antama arvo*/
xDps = (int)(xOut_dps - xOff_dps);
/*****************************************************************************/
/*Luetaan kulmanopeus Y-akselin osalta*/
yAxisGyroscope(yAxisGyroscopeData);
//Muutetaan MSB ja LSB 16bittiseksi arvoksi
yOut16bit_dps = ((short)(yAxisGyroscopeData[0] << 8 | yAxisGyroscopeData[1]));
//Muutetaan arvo todelliseen arvoon eli DPS (degrees per second)
//(+-2000dps) datalehden sivu 23, Taulukko 3)
yOut_dps = ((float) yOut16bit_dps * 0.07);
/*Vähennetään mitatusta arvosta kalibroinnin antama arvo*/
yDps = (int)(yOut_dps - yOff_dps);
/*****************************************************************************/
/*Luetaan kulmanopeus Z-akselin osalta*/
zAxisGyroscope(zAxisGyroscopeData);
//Muutetaan MSB ja LSB 16bittiseksi arvoksi
zOut16bit_dps = ((short)(zAxisGyroscopeData[0] << 8 | zAxisGyroscopeData[1]));
//Muutetaan arvo todelliseen arvoon eli DPS (degrees per second)
//(+-2000dps) datalehden sivu 23, Taulukko 3)
zOut_dps = ((float) zOut16bit_dps * 0.07);
/*Vähennetään mitatusta arvosta kalibroinnin antama arvo*/
zDps = (int)(zOut_dps - zOff_dps);
}
```

Koodi 3. Kulmanopeusarvojen prosessointi.

5.4.5 Liikeanturin kalibrointi

Kaikilla antureilla on aina mittausepätarkkuutta. Mittausepätarkkuuden suuruus saadaan selville anturivalmistajan datalehdestä. Mittausepätarkkuuteen vaikuttaa mm. lämpötila ja kohina. Lisäksi mittausepätarkkuutta voi syntyä, kun anturi liitetään laitteen kytkentään fyysisesti. Mittausepätarkkuus tulee ottaa huomioon anturin mittaustuloksissa. Tästä syystä työssä käytetty kiihtyvyys- ja kulmanopeusanturi kalibroitiin, jotta mittaustavoitteisiin päästäisiin.

Liikeanturi lataa automaattisesti käynnistyksen yhteydessä FLASH-muistista tehtaalla suoritetut kalibrointiarvot, jotka huomioidaan rekistereistä luettaviin mittaustuloksiin. Valmiit kalibrointiarvot eivät kuitenkaan huomio laitteen fyysisestä kytkennästä aiheutuvaa mittausepätarkkuutta. Tästä syystä liikeanturille tehtiin yksinkertainen kalibrointi, jolla pystyttiin poistamaan anturin kytkentään juottamisen yhteydessä tullut mittausepätarkkuus. Liikeanturin kalibrointi suoritettiin heti laitteen käynnistyessä, laitteiston initialisoinnin jälkeen. Kalibroinnin ollessa kokonaisuudessaan valmis, sytytettiin merkiksi kaikki 24 lediä sekunnin ajaksi. Laitteen tulee olla paikallaan ohjelmiston suorittaessa kalibrointia, ettei laitteeseen kohdistu kalibroinnin aikana liikettä, joka johtaisi virheellisiin kalibrointiarvioihin ja näin mittaustuloksiin.

Liitteessä 1 on esitetty kokonaisuudessaan kiihtyvyys- ja kulmanopeusanturin kalibrointi. Kalibrointiarvojen prosessointiin käytettiin silmukoita, jotta yksittäiset mahdolliset virheelliset mittaustulokset eivät vaikuttaneet saatuihin kalibrointiarvoihin. Kiihtyvyysanturin kalibrointiarvo saatiin laskemalla kokonaiskiihtyvyys kaavan 7 mukaan. Saatu kalibrointiarvo jaettiin silmukan kierroksien lukumäärällä, jota seurasi Maan vetovoiman aiheuttaman staattisen voiman vähentäminen. Kulmanopeusanturin kalibrointi suoritettiin akselikohtaisesti, koska työssä tutkittiin laitteen mahdollista kulmanopeutta akseleittain.

Kiihtyvyys- ja kulmanopeusanturi voidaan ohjelmoida kolmeen eri tilaan: alhainen virrantila (low-power), normaali (normal) tai korkea suorituskyky (high-performance). Tässä työssä kiihtyvyys- ja kulmanopeusanturi ohjelmoitiin korkea suorituskykytilaan. Kuvasta 25 nähdään kiihtyvyys- ja kulmanopeusanturin kohinan tiheysspektri korkea suorituskykytilassa. Kiihtyvyysanturin kohinan tiheysspektrin suuruus riippuu kiihtyvyyden mittausrajasta (NXP Semiconductors 2007).

Kiihtyvyys- ja kulmanopeusanturin kohinan RMS-arvo (tehollisarvo) lasketaan alla olevalla kaavalla:

$$Kohina = kohinan\ tiheysspektri \times \sqrt{kaistanleveys \times 1{,}6} \quad (8)$$

Kulmanopeusanturin näytteenantonopeus (ODR) ohjelmoitiin työssä tasoon 104 Hz. Kyseisessä tasossa on kulmanopeusanturin kaistanleveys 33 Hz. Kiihtyvyysanturin kaistanleveys saadaan jakamalla kiihtyvyysanturin näytteenantonopeus kahdella. Kiihtyvyysanturin mittaaman yhden akselin kiihtyvyyden mittausrajaksi ohjelmoitiin $\pm 2\ g$ ja näytteenantonopeudeksi 6664 Hz. Näin kaavan 8 mukaan kiihtyvyysanturin kohinan suuruudeksi saatiin $0{,}005476\ g$ eli $0{,}0537\ \mathrm{m/s^2}$ ja kulmanopeusanturin $0{,}0276\ \mathrm{dps}$. Saadusta tuloksesta voidaan johtaa yksittäisen mittaustuloksen virheen suuruus. Kohinan tehollisarvon ollessa $0{,}005476\ g$ on yksittäisen mittauksen virhe normaalijakautunut keskihajonnalla 0,005476 eli 99,9 % kaikista suoritetuista mittauksista on 3,3 keskihajonnan sisällä keskiarvosta. Kaavan 9 mukaan on siis 0,1 % mittaustapahtumien tuloksista virheellisiä kaavasta saadun tuloksen verran.

$$0{,}005476\ g \times 3{,}3 = 0{,}018071\ g \quad (9)$$

Työn tuloksen tavoitteena oli tutkia jatkuvaa kiihtyvyyttä välillä $-0{,}01\ g\ -\ -0{,}23\ g$. Kaavan perusteella, vaikka laite olisi paikallaan, voidaan kokonaiskiihtyvyyden suuruudeksi mitata ennemmin tai myöhemmin mittaustavoitteen mukainen kiihtyvyys. Työn tavoitteeseen pääsemiseksi tehtiin sulautettuun ohjelmistoon ehtolausekkeet, joilla tutkittiin useaa peräkkäistä kokonaiskiihtyvyyden suuruutta. Mittaustuloksen ollessa siis virheellinen, ei yksittäistä mittaustavoitteen täyttävää kiihtyvyyttä indikoitu ehtolausekkeiden ansiosta.

Lämpötilan muutoksesta johtuvaa mittausepätarkkuutta ei huomioitu tässä työssä, koska muutoksesta johdettava mittausepätarkkuus on erittäin vähäinen, eikä näin vaikuta laitteen toimintaan oleellisesti. Kiihtyvyysanturin näytteenantonopeus ohjelmoitiin mahdollisimman suureksi, että koodin 2 suoritusaika voitiin minimoida.

Kalibroinnin suorittamisen jälkeen, oli jokaisen kulmanopeusanturin akselin mittaustulos laitteen ollessa paikallaan 0 dps. Kokonaiskiihtyvyyden mittaustulos saatiin myös kalibroitua erittäin lähelle 0 g, laitteen ollessa paikallaan. Kiihtyvyysanturin kohinan suuruus

vaikutti kuitenkin kiihtyvyysanturin mittaustuloksiin sen verran, että mittaustuloksista johdettua kokonaiskiihtyvyyttä olisi voitu täysin kalibroida stabiiliksi.

Symbol	Parameter	Test conditions	Min.	Typ.(1)	Max.	Unit
Rn	Rate noise density in high-performance mode(6)			3.8		mdps/√Hz
RnRMS	Gyroscope RMS noise in normal/low-power mode(7)			75		mdps
An	Acceleration noise density in high-performance mode(8)	FS = ±2 *g*		75		µg/√Hz
		FS = ±4 *g*		80		
		FS = ±8 *g*		90		
		FS = ±16 *g*		130		

Kuva 25. Liikeanturin kohinan tiheysspektri (STMicroelectronics 2017).

5.4.6 I/O-porttien initialisointi

I/O-porttien initialisointia varten tehtiin jokaiselle ledille makrot. Ledien makrot ovat esitetty kokonaisuudessaan liitteessä 1. Makroilla voitiin kytkeä ledejä päälle ja pois päältä sekä lisäksi määritellä, että kyseiset I/O-portit ovat ulostuloportteja.

Koodin 4 mukainen funktio luotiin ledSetup.c-lähdekooditiedostoon. Kyseinen funktio asettaa I/O-portit ulostuloporteiksi, jotka ovat kytketty indikoimiseen tarkoitettuihin ledeihin. GPIOSetDir-funktio on ohjelmistokirjaston sisältämä valmis funktio, jolla määritetään I/O-portti ulos- tai sisääntuloportiksi. Kyseisellä funktiolla on kolme parametria. Ensimmäinen parametri on portin ryhmän numero, toinen parametri vastaa portin numeroa ja kolmannella parametrilla määritellään portti ulos- tai sisääntuloportiksi. Kolmannen parametrin arvon ollessa 1 asetetaan portti ulostuloportiksi ja vastaavasti sisääntuloportiksi arvon ollessa 0. Osa I/O-porteista pitää määritellä funktionallisuudeltaan vastamaan yleiskäyttöistä ulos- tai sisääntuloporttia. Tämä tapahtuu muokkaamalla portin rekisteriä. Aluksi resetoidaan kyseinen portti, jonka jälkeen määritellään portti I/O-portiksi. Rekisterin muokkaamisen jälkeen voidaan portti asettaa ulostuloportiksi käyttämällä GPIOSetDir-funktiota. Koodin 4 funktiossa, ledien initialisoinnin lisäksi määriteltiin I²C-väylän SA0-pinni. Portti asetettiin ulostuloportiksi, jonka jälkeen porttiin kirjoitettiin 0 V:n

jännite. 0 V:n jännite ulostuloportissa määrittää LSM6DSM-liikeanturin osoitteen viimeisen bitin arvoksi 0.

```
/*Function initializing Led's as outputs when program starts*/
void setLedPinsOutput(){
/*Set LED'X'_L's to outputs*/
GPIOSetDir( LED1_L_ON );    /*LED 1L*/
GPIOSetDir( LED2_L_ON );    /*LED 2L*/
GPIOSetDir( LED3_L_ON );    /*LED 3L*/
GPIOSetDir( LED4_L_ON );    /*LED 4L*/
GPIOSetDir( LED5_L_ON );    /*LED 5L*/
GPIOSetDir( LED6_L_ON );    /*LED 6L*/
GPIOSetDir( LED7_L_ON );    /*LED 7L*/
GPIOSetDir( LED8_L_ON );    /*LED 8L*/
GPIOSetDir( LED9_L_ON );    /*LED 9L*/
/*We need to reset PIO0_11 -> PIO0_13 manually, then assign it to be
*PIO -port instead of ex. TDI, TMS... and after that set it output -pin*/
LPC_IOCON->TDI_PIO0_11 &= ~(0x07);
LPC_IOCON->TDI_PIO0_11 |= 0x01;
GPIOSetDir( LED10_L_ON );  /*LED 10L*/
LPC_IOCON->TMS_PIO0_12 &= ~(0x07);
LPC_IOCON->TMS_PIO0_12 |= 0x01;
GPIOSetDir( LED11_L_ON );  /*LED 11L*/
LPC_IOCON->TDO_PIO0_13 &= ~(0x07);
LPC_IOCON->TDO_PIO0_13 |= 0x01;
GPIOSetDir( LED12_L_ON );  /*LED 12L*/
LPC_IOCON->TRST_PIO0_14 &= ~(0x07);
LPC_IOCON->TRST_PIO0_14 |= 0x01;
GPIOSetDir( LED1_R_ON );    /*LED 12L*/
/*****************************************************************/
/*Set LED'X'_R's to outputs*/
GPIOSetDir( LED2_R_ON );    /*LED 2L*/
GPIOSetDir( LED3_R_ON );    /*LED 3L*/
GPIOSetDir( LED4_R_ON );    /*LED 4L*/
GPIOSetDir( LED5_R_ON );    /*LED 5L*/
GPIOSetDir( LED6_R_ON );    /*LED 6L*/
GPIOSetDir( LED7_R_ON );    /*LED 7L*/
GPIOSetDir( LED8_R_ON );    /*LED 8L*/
GPIOSetDir( LED9_R_ON );    /*LED 9L*/
GPIOSetDir( LED10_R_ON );  /*LED 7L*/
GPIOSetDir( LED11_R_ON );  /*LED 8L*/
GPIOSetDir( LED12_R_ON );  /*LED 9L*/
GPIOSetDir( I2C_SA0 );
GPIOSetBitValue( 0, 21, 0);  //I2C address select!
}
```

Koodi 4. I/O-porttien initialisointi.

5.5 Testaus

Laitteiston mittaustavoite ja muut sulautetun ohjelmiston toiminnan kannalta merkittävien muuttujien raja-arvot määriteltiin testaamalla laitetta kohdeympäristössä. Laitteistoa testattiin hyvin yksinkertaisella tavalla laitteen reunaehtojen puitteissa. Mittaustulokset saatiin tulostamalla testattavan muuttujan arvoa printf-funktiolla.

Testaaminen aloitettiin selvittämällä laitteiston kokonaiskiihtyvyyden mittaustavoite, minkä sulautetun ohjelmiston tulee indikoida käyttäjälle. Kokonaiskiihtyvyyden ollessa vektori ja vektorin saamiseksi käytettiin kaavaa 7, oli kyseinen mittaustulos positiivinen, vaikka työn mittaustavoite oli tutkia negatiivista kiihtyvyyttä. Positiivinen kiihtyvyys eli laitteen nopeuden kasvaminen stabiilissa kohdeympäristössä oli kuitenkin suhteessa niin pientä negatiiviseen kiihtyvyyteen verrattuna, että laitteeseen toteutettu sulautettu ohjelmisto ei tätä indikoinut. Kokonaiskiihtyvyyden mittaustavoite johdettiin laitteen liikkeen eri hidastumistilanteissa eli hitaassa, normaalissa ja äkillisessä hidastumisessa. Laitteen hitaassa hidastumisliikkeessä oli mitattu arvo $0{,}01\ g$, normaalissa hidastumisessa $0{,}10\ g$ ja äkillisessä hidastumisessa arvoksi mitattiin $0{,}23\ g$. Myös suurempia mittaustuloksia saatiin äkillisessä hidastumisessa, mutta $0{,}23\ g$ oli yleisin mittaustulos ja siksi mittaustavoitteen viimeiseksi raja-arvoksi määriteltiin kyseinen kokonaiskiihtyvyys. Laiteen tuli kuitenkin indikoida myös suurempia mittaustuloksia kuin $0{,}23\ g$. Mikäli mitattu kokonaiskiihtyvyys ylitti kyseisen arvon ja täytti sulautetun ohjelmiston reunaehdot, laite indikoi kyseisen kiihtyvyyden kytkemällä päälle kaikki laitteen ledit ja tämän jälkeen aloitti pois kytkemisen porrastetusti, mikäli nykyinen mitattu kokonaiskiihtyvyys oli pienempi kuin edellinen.

Mittaustuloksien perusteella johdettiin kuvion 3 mukainen lineaarinen funktio. Lineaarinen funktio muodostettiin asettamalla ensimmäiseksi arvoksi laitteen hitaassa hidastumisliikkeessä mitattu arvo eli $0{,}01\ g$ ja viimeiseksi arvoksi määriteltiin $0{,}23\ g$. Lineaarinen funktio muodostui 12 eri arvosta. Näin jokaiselle laitteen R ja L ryhmän ledille saatiin kuvaajan perusteella asetettua ledien päälle kytkemisen raja-arvo.

Sulautetun ohjelmiston laskurimuuttujien raja-arvot eli ledien päälle kytkemiseen tarvittavat laskurisummat johdettiin laitteen eri hidastumistilanteista. Mittaustuloksista laskettiin summa, kuinka kauan kokonaiskiihtyvyys pysyi kyseisen raja-arvon yläpuolella.

Kolme yleisintä saatua raja-arvoa, jonka yläpuolella kokonaiskiihtyvyyden mittaustulos pysyi laitteen eri liikkeen hidastumistilanteissa, olivat $0{,}01\ g$, $0{,}025\ g$ ja $0{,}05\ g$. Mittaustuloksista johdettiin sulautetun ohjelmiston ehtolausekkeille kyseiset kokonaiskiihtyvyyden raja-arvot. $0{,}01\ g$ ja $0{,}025\ g$ suuruisten kokonaiskiihtyvyyksien osalta, ei mitattu kokonaiskiihtyvyys kuitenkaan aina pysynyt kyseisen raja-arvon yläpuolella. Tästä syystä tehtiin exception-totuusarvomuuttujat, joiden avulla tarkkailtiin, nouseeko mitattu kokonaiskiihtyvyys takaisin kyseisen raja-arvon yläpuolelle vai ei.

Kuvio 3. Kokonaiskiihtyvyyden mittaustulos.

Kokonaiskiihtyvyyden mittaustavoite johdettiin stabiilissa kohdeympäristössä. Laitteen kohdeympäristö saattoi kuitenkin olla epästabiili, joten laitetta testattiin myös kyseisessä ympäristössä. Epästabiili kohdeympäristö sisälsi paljon laitteeseen kohdistuvaa tärinää ja satunnaisia suuria vertikaalisia kiihtyvyyden muutoksia, jotka johtuivat pompuista tai muista vastaavista äkillisen kiihtyvyyden muutoksen aiheuttavista tekijöistä. Laitteen mittaustuloksista epästabiilissa kohdeympäristössä, johdettiin raja-arvot kulmanopeusanturin eri akseleille ja äkilliselle kokonaiskiihtyvyyden muutokselle. Kokonaiskiihtyvyyden eroavaisuuden ollessa suurempi tai yhtä suuri kuin $0{,}6\ g$ edelliseen mitattuun arvoon nähden, oli laitteistoon kohdistunut äkillinen kiihtyvyyden muutos. Kyseistä tulosta käy-

tettiin triggerLevel-muuttujan raja-arvona. Laitteeseen kohdistuva tärinä havaittiin kulmanopeusanturin eri akselien mittaustuloksissa. Tärinän tuoma kiihtyvyyden muutos saatiinkin siis suljettua pois käyttämällä kulmanopeusanturilta saatavia tuloksia. Kulmanopeusanturin eri akseleille määriteltiin raja-arvoksi $\pm 7\ \mathrm{dps}$.

Laitteistoon kohdistui myös kulmakiihtyvyyttä laitteen kääntyessä, joka havaittiin kulmanopeusanturin akselien mittaustuloksissa. Kulmakiihtyvyyden aiheuttama muutos kokonaiskiihtyvyyden mittaustulokseen huomioitiin näin kulmanopeusanturilla. Edellistä määriteltyä raja-arvoa eli $\pm 7\ \mathrm{dps}$ käytettiin myös kulmakiihtyvyyden pois sulkemiseen. Raja-arvon määrittämisessä tuli kuitenkin huomioida, että kokonaiskiihtyvyyttä tulisi silti indikoida, vaikka laitteeseen kohdistuisi pientä tärinää tai kulmakiihtyvyyttä. Mikäli raja-arvo olisi määritelty liian alas, ei laite olisi näin toiminut lainkaan epästabiilissa kohdeympäristössä. Lisäksi kulmanopeusanturin akseleille määriteltiin toinen raja-arvo $\pm 20\ \mathrm{dps}$, joka asetettiin dpsLvl-muuttujan raja-arvoksi. Mikäli kulmanopeusanturin akselin mittaustulos oli suurempi kuin kyseinen raja-arvo, oli laitteen kulmakiihtyvyys erittäin suuri ja kokonaiskiihtyvyyttä ei indikoitu ennen kuin kokonaiskiihtyvyyden mittaustulos oli palautunut takaisin stabiiliksi.

Laitteen sulautetun ohjelmiston ollessa kokonaisuudessaan valmis testattiin laitetta eri kohdeympäristöissä. Laite täytti vaatimusmäärittelyn mukaiset ehdot, kun laitetta testattiin stabiilissa kohdeympäristössä. Siirryttäessä kuitenkin yhä epästabiilimpaan ympäristöön laite indikoi hetkittäin satunnaisia, ei haluttuja kokonaiskiihtyvyyksiä johtuen kohdeympäristön olosuhteista. Liikkeen hidastuminen indikoitiin edelleen vaatimusmäärittelyn mukaisesti, vaikka laitetta käytettiinkin epästabiilissa ympäristössä. Epästabiilista kohdeympäristöstä johtuvien häiriötekijöiden vuoksi, ei projektin vaatimusmäärittelyn mukaista laitetta voitu siltä osin kehittää täysin toimivaksi, että laitteeseen kohdistuvien satunnaisten kiihtyvyyksien indikointi olisi voitu rajata täysin laitteiston toiminnan ulkopuolelle. Työn tuloksena saatiin kuitenkin kehitettyä laite prototyypiksi, jolla voitiin esittää suunnitelman mukaista laitteen toimintaperiaatetta.

Varsinaisia mittaustuloksia valmiin laitteen testaamisesta ei voida työssä esitellä, koska laitteen mittaustuloksien lukemiseen käytettiin LPCXpresso-ohjelmointiympäristöä ja printf-funktiota. Printf-funktio hidasti laitteen sulautetun ohjelmiston suorittamista merkittävästi, joten mittaustuloksien saaminen ja samalla laitteen funktionaalisen toiminnan testaaminen samanaikaisesti ei ollut mahdollista.

6 YHTEENVETO

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa sulautettu ohjelmisto laitteistoon, jonka avulla voitaisiin indikoida käyttäjälle laitteeseen kohdistuvaa negatiivista kiihtyvyyttä eli liikkeen hidastumista LSM6DSM-liikeanturilla. Työn kohdeympäristön ollessa haastava kohdistuu laitteeseen äkillisiä voimia, jotka vaikuttavat suoraan anturin lukemiin. Kaikki kohdeympäristön tuoma häiriö tulee ottaa huomioon ja ymmärtää sen vaikutus anturin toiminnassa, ennen kuin johdettuja tuloksia tulee indikoida.

Työn tuloksena syntyi prototyyppi, joka täytti työn vaatimusmäärittelyn mukaiset kriteerit käytettäessä laitetta häiriöttömissä olosuhteissa. Prototyyppiä käytettäessä epästabiilissa kohdeympäristössä laite indikoi harvakseen satunnaisia, ei haluttuja kiihtyvyyksiä laitteeseen kohdistuvien äkillisten voimien vuoksi. Liikkeen hidastuminen voitiin kuitenkin indikoida vaatimusmäärittelyn mukaisesti, vaikka laitetta käytettiin häiriöllisissä olosuhteissa. Työn kokonaisuutena voidaan katsoa, että työn tuloksena syntyneellä prototyypillä voidaan esittää suunnitelman mukaista laitteen toimintaperiaatetta. Lisäksi voidaan todeta, että liikeanturin käyttäminen häiriöllisissä olosuhteissa vaatii paljon testaamista ja johdettuja mittaustuloksia, joiden avulla voidaan rajata laitteen funktionaalista toimintaa.

Laitteiston kehitystä tullaan jatkamaan erilaista lähestymistapaa käyttäen. Kiihtyvyyden ollessa vektorisuure, joka kuvaa nopeuden muutosta tietyssä ajassa, voidaan laitteeseen kohdistuva negatiivinen kiihtyvyys indikoida luomalla laitteeseen rajapinta, jolla saadaan selville laitteen nopeus. Näin ollen haastavan kohdeympäristön tuoma häiriö ei vaikuta laitteen toimintaan, kun kiihtyvyyden indikoimiseen käytetään saatua nopeutta kiihtyvyysanturin sijaan.

LÄHTEET

Bell-Labs 2003. The Development of the C Language*. Viitattu 16.3.2018 https://www.bell-labs.com/usr/dmr/www/chist.html.

Dadafshar, M. 2014. Accelerometer and gyroscopes sensors: operation, sensing, and applications. Viitattu 13.3.2018 https://pdfserv.maximintegrated.com/en/an/AN5830.pdf.

Dimension Engineering LLC 2018. A beginner's guide to accelerometers. Viitattu 12.3.2018 https://www.dimensionengineering.com/info/accelerometers.

Embedded Artists AB 2012. LPCXpresso LPC11U14 rev A. Viitattu 21.3.2018 https://www.embeddedartists.com/sites/default/files/docs/schematics/LPCXpressoLPC11U14revA.pdf.

Hutasu 2017. Sarjaliikenne – I2C. Viitattu 2.3.2018 https://www.hutasu.net/elektroniikka/sulautettu-elektroniikka/sarjaliikenne-i2c/.

LPC Tools 2014. LPC11U14 LPCXpresso board. Viitattu 16.3.2018 http://www.lpctools.com/lpc11u14.lpcxpresso.aspx.

NXP Semiconductors 2007. Accelerometer Terminology Guide. Viitattu 2.4.2018 http://cache.freescale.com/files/sensors/doc/support_info/SENSORTERMSPG.pdf.

NXP Semiconductors 2013. LPCXpresso IDE User Guide. Viitattu 16.3.2018 https://www.mouser.com/pdfdocs/LPCXpresso_IDE_User_Guide.pdf.

NXP Semiconductors 2014. LPC11U1x. Viitattu 28.2.2018 https://www.nxp.com/docs/en/data-sheet/LPC11U1X.pdf.

NXP Semiconductors 2016. UM10462 LPC11U3x/2x/1x User manual. Viitattu 2.3.2018 https://www.nxp.com/docs/en/user-guide/UM10462.pdf.

NXP Semiconductors 2018. OM13014: LPCXpresso Board for LPC11U14. Viitattu 16.3.2018 https://www.nxp.com/support/developer-resources/hardware-development-tools/lpcxpresso-boards/lpcxpresso-board-for-lpc11u14:OM13014.

Robot Electronics 2018. Using the I2C Bus. Viitattu 2.3.2018 http://www.robot-electronics.co.uk/i2c-tutorial.

Sensorwiki 2016. Gyroscope. Viitattu 13.3.2018 http://sensorwiki.org/doku.php/sensors/gyroscope.

STMicroelectronics 2017. LSM6DSM. Viitattu 2.3.2018 http://www.st.com/content/ccc/resource/technical/document/datasheet/76/27/cf/88/c5/03/42/6b/DM00218116.pdf/files/DM00218116.pdf/jcr:content/translations/en.DM00218116.pdf.

Trusov, A. 2011. Overview of MEMS Gyroscopes: History, Principles of Operations, Types of Measurements. Viitattu 13.3.2018 http://www.alexandertrusov.com/uploads/pdf/2011-UCI-trusov-whitepaper-gyros.pdf.

Williams, M. 2017. Converting values from an Accelerometer to G. Viitattu 15.3.2018 http://ozzmaker.com/accelerometer-to-g/.

Liite 1. Sulautetun ohjelmiston lähdekoodi.

X-akselin kiihtyvyysarvojen lukeminen.

```
/*Function to read x-Axis from the accelerometer*/
void xAxisAccelerometer(unsigned char *xAxisArray){

/*READ X LSB*/

//Set start bit
LPC_I2C->CONSET       = (1<<5);          //Set Startbit

while(!(LPC_I2C->CONSET & (1<<3)));      //Wait for interrupt to be set
LPC_I2C->DAT         = writeAddress;     //Device address + Write
LPC_I2C->CONCLR     = (1<<5);            //Reset STA
LPC_I2C->CONCLR     = (1<<3);            //Reset Interrupt

//Transmit registry address to read from
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->DAT        = OUTX_L_XL;
LPC_I2C->CONCLR     = (1<<3);

//Set start bit
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->CONSET     = (1<<5);

//Device address + Read
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->DAT        = readAddress;        //Device address + Read

//Wait for last i2c operation to execute
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<5);             //Clear Startbit
LPC_I2C->CONCLR     = (1<<3);

//Read X-axis LSB
while(!(LPC_I2C->CONSET & (1<<3)));
xAxisArray[1] = LPC_I2C->DAT;
LPC_I2C->CONSET             = (1<<2);     // ACK read data
LPC_I2C->CONCLR     = (1<<3);

//Set stop bit and clear all bits
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR   = (1<<5);
LPC_I2C->CONSET   = (1<<4);
LPC_I2C->CONCLR   = (1<<3);
LPC_I2C->CONCLR   = (1<<2);

/*****************************************************************************/

*READ X MSB*/

//Set start bit
LPC_I2C->CONSET       = (1<<5);          //Set Startbit

while(!(LPC_I2C->CONSET & (1<<3)));      //Wait for interrupt to be set
LPC_I2C->DAT         = writeAddress;     //Device address + Write
LPC_I2C->CONCLR     = (1<<5);            //Reset STA
LPC_I2C->CONCLR     = (1<<3);            //Reset Interrupt

//Transmit registry address to read from
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->DAT        = OUTX_H_XL;
LPC_I2C->CONCLR     = (1<<3);

//Set start bit
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->CONSET     = (1<<5);
```

```
//Device address + Read
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->DAT        = readAddress;        //Device address + Read

//Wait for last i2c operation to execute
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<5);             //Clear Startbit
LPC_I2C->CONCLR     = (1<<3);

//Read X-axis MSB
while(!(LPC_I2C->CONSET & (1<<3)));
xAxisArray[0] = LPC_I2C->DAT;
LPC_I2C->CONSET             = (1<<2);     // ACK read data
LPC_I2C->CONCLR     = (1<<3);

//Set stop bit and clear all bits
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR   = (1<<5);
LPC_I2C->CONSET   = (1<<4);
LPC_I2C->CONCLR   = (1<<3);
LPC_I2C->CONCLR   = (1<<2);

}
```

X-akselin kulmanopeusarvojen lukeminen.

```
/*Function to read x-Axis from the Gyroscope*/
void xAxisGyroscope(unsigned char *xAxisArray){

/*READ X LSB*/

//Set start bit
LPC_I2C->CONSET       = (1<<5);                         //Set Startbit

while(!(LPC_I2C->CONSET & (1<<3)));                     //Wait for interrupt to be set
LPC_I2C->DAT         = writeAddress;                    //Device address + Write
LPC_I2C->CONCLR     = (1<<5);                           //Reset STA
LPC_I2C->CONCLR     = (1<<3);                           //Reset Interrupt

//Transmit registry address to read from
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->DAT        = OUTX_L_G;
LPC_I2C->CONCLR     = (1<<3);

//Set start bit
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->CONSET     = (1<<5);

//Device address + Read
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->DAT        = readAddress;                      //Device address + Read

//Wait for last i2c operation to execute
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<5);                           //Clear Startbit
LPC_I2C->CONCLR     = (1<<3);

//Read X-axis LSB
while(!(LPC_I2C->CONSET & (1<<3)));
xAxisArray[1] = LPC_I2C->DAT;
LPC_I2C->CONSET             = (1<<2);                   // ACK read data
LPC_I2C->CONCLR     = (1<<3);

//Set stop bit and clear all bits
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR   = (1<<5);
LPC_I2C->CONSET   = (1<<4);
LPC_I2C->CONCLR   = (1<<3);
```

```
LPC_I2C->CONCLR   = (1<<2);

/*****************************************************************************/

*READ X MSB*/

//Set start bit
LPC_I2C->CONSET       = (1<<5);                         //Set Startbit

while(!(LPC_I2C->CONSET & (1<<3)));                     //Wait for interrupt to be set
LPC_I2C->DAT         = writeAddress;                    //Device address + Write
LPC_I2C->CONCLR     = (1<<5);                           //Reset STA
LPC_I2C->CONCLR     = (1<<3);                           //Reset Interrupt

//Transmit registry address to read from
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->DAT        = OUTX_H_G;
LPC_I2C->CONCLR     = (1<<3);

//Set start bit
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->CONSET     = (1<<5);

//Device address + Read
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->DAT        = readAddress;                      //Device address + Read

//Wait for last i2c operation to execute
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<5);                           //Clear Startbit
LPC_I2C->CONCLR     = (1<<3);

//Read X-axis MSB
while(!(LPC_I2C->CONSET & (1<<3)));
xAxisArray[0] = LPC_I2C->DAT;
LPC_I2C->CONSET            = (1<<2);                    // ACK read data
LPC_I2C->CONCLR     = (1<<3);

//Set stop bit and clear all bits
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR   = (1<<5);
LPC_I2C->CONSET   = (1<<4);
LPC_I2C->CONCLR   = (1<<3);
LPC_I2C->CONCLR   = (1<<2);


}
```

accelerometer.h

```
#ifndef ACCELEROMETER_H_
#define ACCELEROMETER_H_

void I2CInit( void );
void sendData(unsigned int address, unsigned int data);

void xAxisAccelerometer(unsigned char *xAxisArray);
void yAxisAccelerometer(unsigned char *yAxisArray);
void zAxisAccelerometer(unsigned char *zAxisArray);

void xAxisGyroscope(unsigned char *xAxisArray);
void yAxisGyroscope(unsigned char *yAxisArray);
void zAxisGyroscope(unsigned char *zAxisArray);

unsigned char readStatusRegisterAcc (unsigned char accStatus);
unsigned char readStatusRegisterGyro(unsigned char gyroStatus);

void _delay_ms (uint16_t ms);

/*****************************************************************************
************/

/*Read and Write addresses of the sensor LSM6DSM (SA0 = 0V)*/
#define readAddress 0xD5
#define writeAddress 0xD4

/*****************************************************************************
************/

/*Control registers of Accelerometer and Gyroscope*/
#define CTRL1_XL 0x10 // Linear acceleration sensor control register
#define CTRL2_G 0x11  // Angular rate sensor control register
#define CTRL3_C 0x12  // Register that contains Block Data Update
// Output registers not updated until MSB and LSB have been read

#define CTRL8_XL 0x17

#define INT1_CTRL 0x0D
#define STATUS_REG 0x1E

/*****************************************************************************
************/

/*X -axis of accelerometer*/
#define OUTX_L_XL 0x28 // Linear acceleration sensor X-axis output register (Least
significant part)
#define OUTX_H_XL 0x29 // Linear acceleration sensor X-axis output register (Most
significant part)

/*Y -axis of accelerometer*/
#define OUTY_L_XL 0x2A // Linear acceleration sensor Y-axis output register (Least
significant part)
#define OUTY_H_XL 0x2B // Linear acceleration sensor Y-axis output register (Most
significant part)

/*Z -axis of accelerometer*/
#define OUTZ_L_XL 0x2C // Linear acceleration sensor Z-axis output register (Least
significant part)
#define OUTZ_H_XL 0x2D // Linear acceleration sensor Z-axis output register (Most
significant part)

/*****************************************************************************
************/

/*X -axis of Angular rate sensor*/
#define OUTX_L_G 0x22 // Angular rate sensor X-axis output register (Least significant
part)
#define OUTX_H_G 0x23 // Angular rate sensor X-axis output register (Most significant
part)
```

```
/*Y -axis of Angular rate sensor*/
#define OUTY_L_G 0x24 // Angular rate sensor Y-axis output register (Least significant
part)
#define OUTY_H_G 0x25 // Angular rate sensor Y-axis output register (Most significant
part)

/*Z -axis of Angular rate sensor*/
#define OUTZ_L_G 0x26 // Angular rate sensor Z-axis output register (Least significant
part)
#define OUTZ_H_G 0x27 // Angular rate sensor Z-axis output register (Most significant
part)

/*****************************************************************************
*************/

#define I2CONCLR_AAC        (0x1<<2)    /* I2C Control clear Register */
#define I2CONCLR_SIC        (0x1<<3)
#define I2CONCLR_STAC       (0x1<<5)
#define I2CONCLR_I2ENC      (0x1<<6)

#endif /* ACCELEROMETER_H_ */
```

ledsetup.h

```
#ifndef LEDSETUP_H_
#define LEDSETUP_H_

void setLedPinsOutput();
void turnLedsOff();
void turnLedsOn(float acceleration);
void turnLedsOnAfterCalibration();
void initializeAccelerationArray();

/*I2C SA0*/
#define I2C_SA0 0, 21, 1


/*DEFINE LED1_L ON's*/
#define LED1_L_ON 1, 24, 1
#define LED2_L_ON 0, 6, 1
#define LED3_L_ON 0, 7, 1
#define LED4_L_ON 1, 28, 1
#define LED5_L_ON 1, 31, 1
#define LED6_L_ON 1, 21, 1
#define LED7_L_ON 0, 8, 1
#define LED8_L_ON 0, 22, 1
#define LED9_L_ON 1, 29, 1
#define LED10_L_ON 0, 11, 1
#define LED11_L_ON 0, 12, 1
#define LED12_L_ON 0, 13, 1


/*DEFINE LED1_R ON's*/
#define LED1_R_ON 0, 14, 1
#define LED2_R_ON 1, 13, 1
#define LED3_R_ON 1, 14, 1
#define LED4_R_ON 1, 22, 1
#define LED5_R_ON 0, 16, 1
#define LED6_R_ON 0, 23, 1
#define LED7_R_ON 1, 15, 1
#define LED8_R_ON 0, 17, 1
#define LED9_R_ON 0, 18, 1
#define LED10_R_ON 0, 19, 1
#define LED11_R_ON 1, 16, 1
#define LED12_R_ON 1, 25, 1

/*DEFINE LED1_L OFF's*/
#define LED1_L_OFF 1, 24, 0
#define LED2_L_OFF 0, 6, 0
#define LED3_L_OFF 0, 7, 0
#define LED4_L_OFF 1, 28, 0
#define LED5_L_OFF 1, 31, 0
#define LED6_L_OFF 1, 21, 0
#define LED7_L_OFF 0, 8, 0
#define LED8_L_OFF 0, 22, 0
#define LED9_L_OFF 1, 29, 0
#define LED10_L_OFF 0, 11, 0
#define LED11_L_OFF 0, 12, 0
#define LED12_L_OFF 0, 13, 0

/*DEFINE LED1_R OFF's*/
#define LED1_R_OFF 0, 14, 0
#define LED2_R_OFF 1, 13, 0
#define LED3_R_OFF 1, 14, 0
#define LED4_R_OFF 1, 22, 0
#define LED5_R_OFF 0, 16, 0
#define LED6_R_OFF 0, 23, 0
#define LED7_R_OFF 1, 15, 0
#define LED8_R_OFF 0, 17, 0
#define LED9_R_OFF 0, 18, 0
#define LED10_R_OFF 0, 19, 0
#define LED11_R_OFF 1, 16, 0
#define LED12_R_OFF 1, 25, 0
#endif /* LEDSETUP_H_ */
```

Liikeanturin kalibrointi

```
/*Kiihtyvyysanturin kalibrointi*/
while(accDataCheckCounter < 200){

accDataCheck = readStatusRegisterAcc(accDataCheck);

if(accDataCheck){

//Luetaan kiihtyvyys X-akselilta
xAxisAccelerometer(xAxisAccelerometerData);

//Muutetaan MSB ja LSB 16bittiseksi arvoksi
xOut16bit_g = ((short)(xAxisAccelerometerData[0] << 8 | xAxisAccelerometerData[1]));
//Muutetaan arvo todelliseen arvoon eli g-voimiksi
//(+-2g) datalehden sivu 23, Taulukko 3)
xOff_g = ((float) xOut16bit_g * 0.000061);

//Luetaan kiihtyvyys Y-akselilta
yAxisAccelerometer(yAxisAccelerometerData);

//Muutetaan MSB ja LSB 16bittiseksi arvoksi
yOut16bit_g = ((short)(yAxisAccelerometerData[0] << 8 | yAxisAccelerometerData[1]));
//Muutetaan arvo todelliseen arvoon eli g-voimiksi
//(+-2g) datalehden sivu 23, Taulukko 3)
yOff_g = ((float) yOut16bit_g * 0.000061);

//Luetaan kiihtyvyys Z-akselilta
zAxisAccelerometer(zAxisAccelerometerData);

//Muutetaan MSB ja LSB 16bittiseksi arvoksi
zOut16bit_g = ((short)(zAxisAccelerometerData[0] << 8 | zAxisAccelerometerData[1]));
//Muutetaan arvo todelliseen arvoon eli g-voimiksi
//(+-2g) datalehden sivu 23, Taulukko 3)
zOff_g = ((float) zOut16bit_g * 0.000061);

/*Lasketaan kokonaiskiihtyvyys joka saadaan ko. kaavalla*/
accelerationOffSet += sqrt((xOff_g * xOff_g) + (yOff_g * yOff_g) + (zOff_g * zOff_g));

/*Lisätään laskurin arvoa yhdellä, arvon ollessa 200 poistutaan silmukasta*/
accDataCheckCounter += 1;

            }

}


/*Kulmanopeusanturin kalibrointi*/
while(gyroDataCheckCounter < 500){

/*Luetaan kulmanopeus X-akselin osalta*/
xAxisGyroscope(xAxisGyroscopeData);

//Muutetaan MSB ja LSB 16bittiseksi arvoksi
xOut16bit_dps = ((short)(xAxisGyroscopeData[0] << 8 | xAxisGyroscopeData[1]));
//Muutetaan arvo todelliseen arvoon eli DPS (degrees per second)
//(+-2000dps) datalehden sivu 23, Taulukko 3)
xOff_dps += ((float) xOut16bit_dps * 0.07);

/*Luetaan kulmanopeus Y-akselin osalta*/
yAxisGyroscope(yAxisGyroscopeData);

//Muutetaan MSB ja LSB 16bittiseksi arvoksi
yOut16bit_dps = ((short)(yAxisGyroscopeData[0] << 8 | yAxisGyroscopeData[1]));
//Muutetaan arvo todelliseen arvoon eli DPS (degrees per second)
//(+-2000dps) datalehden sivu 23, Taulukko 3)
yOff_dps += ((float) yOut16bit_dps * 0.07);

/*Luetaan kulmanopeus Z-akselin osalta*/
zAxisGyroscope(zAxisGyroscopeData);
```

```
//Muutetaan MSB ja LSB 16bittiseksi arvoksi
zOut16bit_dps = ((short)(zAxisGyroscopeData[0] << 8 | zAxisGyroscopeData[1]));
//Muutetaan arvo todelliseen arvoon eli DPS (degrees per second)
//(+-2000dps) datalehden sivu 23, Taulukko 3)
zOff_dps += ((float) zOut16bit_dps * 0.07);

/*Lisätään laskurin arvoa yhdellä, arvon ollessa 500 poistutaan silmukasta*/
gyroDataCheckCounter += 1;
}

/*Jaetaan lasketut arvot kalibrointisilmukan kierroksien määrällä*/
xOff_dps /= 500;
yOff_dps /= 500;
zOff_dps /= 500;


/*Jaetaan laskettu arvo kokonaiskiihtyvyyden kalibrointiarvolle kierroksien määrällä*/
accelerationOffSet /= 200;
/*Vähennetään kalibrointiarvosta maanvetovoima eli 1g*/
accelerationOffSet -= 1;

/*Sytytetään kaikki ledit, jolla indikoidaan että kalibrointi on valmis
*Sytyttämistä seuraa ledien sammutus*/
turnLedsOnAfterCalibration();
turnLedsOff();
```

Datan lähettäminen I²C-väylään.

```
/*Send data to the I2C Interface
* Routine: Start, Transmit device address, Transmit Control byte (address),
* Transmit Data (data), Stop
* Parameters: Address where to write, data what to write
* */
void sendData(unsigned int address, unsigned int data){

unsigned int status = 0;

//I2C START TRANSMIT
LPC_I2C->CONSET |= (1<<5);            //set start bit to initiate transmission (datasheet
14.7.1)

do{                                    //wait for start condition to be sent
/*store current state in status (datasheet 14.7.2)
*
*  When the status code is 0xF8,
*  there is no relevant information available and the SI bit is not set.
*  (SI bit is set when the I2C state changes, however, entering state F8
*  does not set the SI bit since there is nothing for an interrupt service
*  to do in that case. */
status = LPC_I2C->STAT & 0xF8;
}while(status != 0x08);                    //compare current state with possible states
(datasheet 14.10.1 table 287)

/*Transmit device address (datasheet 14.7.3)*/
/*Write address of sensor LMS6DSM when SA0 = 0*/
LPC_I2C->DAT        = writeAddress;
/*Clear STA and SI bit in CON register that controls operation of the I2C interface
(datasheet 14.7.6) */
LPC_I2C->CONCLR     = (1<<5); //STA
LPC_I2C->CONCLR     = (1<<3); //SI

//TRANSMIT CONTROL BYTE
//while(LPC_I2C->STAT != 0x18);           //wait for address byte to be sent (datasheet
14.10.1 table 287)
while(!(LPC_I2C->CONSET & (1<<3)));     //wait until SI (Interrupt Flag) is set to 0
LPC_I2C->DAT        = address;         //send data (datasheet 14.7.3)
LPC_I2C->CONCLR     = (1<<3);          //clear SI (datasheet 14.7.6)
```

```
//TRANSMIT DATA BYTE
//while(LPC_I2C->STAT != 0x28);           //wait for address byte to be sent (datasheet
14.10.1 table 287)
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->DAT        = data;            //send data (datasheet 14.7.3)
LPC_I2C->CONCLR     = (1<<3);          //clear SI (datasheet 14.7.6)

//INITIATE STOP CONDITION
//while(LPC_I2C->STAT != 0x28);           //wait for address byte to be sent (datasheet
14.10.1 table 287)
//while((LPC_I2C->CONSET & 0x8) != 0x8); //set STOP bit (sec 14.7.1)
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<5);                                          //STA
LPC_I2C->CONSET     = (1<<4);          //set stop bit (sec 14.7.1)
LPC_I2C->CONCLR     = (1<<3);          //clear SIC (sec 14.7.6)

}
```

Liikeanturin tilarekisterin tarkistaminen.

```
unsigned char readStatusRegisterAcc(unsigned char accStatus){

LPC_I2C->CONSET       = (1<<5);                       //Set Startbit

while(!(LPC_I2C->CONSET & (1<<3)));      //Wait for interrupt to be set
LPC_I2C->DAT         = writeAddress;     //Device address + WRITE
LPC_I2C->CONCLR     = (1<<5);                                       //Reset STA
LPC_I2C->CONCLR     = (1<<3);                                       //Reset interrupt

//Transmit registry address to read from
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->DAT        = STATUS_REG;
LPC_I2C->CONCLR     = (1<<3);

//Set start bit
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->CONSET     = (1<<5);            //Set Startbit

//Device address + Read
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->DAT        = readAddress;       //Device address + Read

//Wait for last i2c operation to execute
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<5);             //Clear Startbit
LPC_I2C->CONCLR     = (1<<3);


while(!(LPC_I2C->CONSET & (1<<3)));

if(LPC_I2C->DAT & (1<<0)){
accStatus = 1;
}
else{
accStatus = 0;
}

LPC_I2C->CONSET            = (1<<2);          // ACK read data
LPC_I2C->CONCLR     = (1<<3);

//Set stop bit and clear all bits
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR   = (1<<5);
LPC_I2C->CONSET   = (1<<4);
LPC_I2C->CONCLR   = (1<<3);
LPC_I2C->CONCLR   = (1<<2);

return accStatus;
}
```

```
unsigned char readStatusRegisterGyro(unsigned char gyroStatus){

LPC_I2C->CONSET       = (1<<5);                         //Set Startbit

while(!(LPC_I2C->CONSET & (1<<3)));      //Wait for interrupt to be set
LPC_I2C->DAT          = writeAddress;     //Device address + WRITE
LPC_I2C->CONCLR     = (1<<5);                                        //Reset STA
LPC_I2C->CONCLR     = (1<<3);                                        //Reset interrupt

//Transmit registry address to read from
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->DAT        = STATUS_REG;
LPC_I2C->CONCLR     = (1<<3);

//Set start bit
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->CONSET     = (1<<5);            //Set Startbit

//Device address + Read
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<3);
LPC_I2C->DAT        = readAddress;       //Device address + Read

//Wait for last i2c operation to execute
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR     = (1<<5);             //Clear Startbit
LPC_I2C->CONCLR     = (1<<3);

while(!(LPC_I2C->CONSET & (1<<3)));

if(LPC_I2C->DAT & (1<<1)){
gyroStatus = 1;
}
else{
gyroStatus = 0;
}
LPC_I2C->CONSET             = (1<<2);          // ACK read data
LPC_I2C->CONCLR     = (1<<3);

//Set stop bit and clear all bits
while(!(LPC_I2C->CONSET & (1<<3)));
LPC_I2C->CONCLR   = (1<<5);
LPC_I2C->CONSET   = (1<<4);
LPC_I2C->CONCLR   = (1<<3);
LPC_I2C->CONCLR   = (1<<2);

return gyroStatus;
}
```

Kiihtyvyyden raja-arvojen määrittäminen.

```
/*Funktio joka asettaa raja-arvot eri kiihtyvyyden asteille*/
void initializeAccelerationArray(){

accelerationRange[0] = rangeSet;
for(int i = 1; i < 12; i++){

accelerationRange[i] = accelerationRange[i-1] + 0.02;
}

}
```